Verteilte Objekte im Internet und Intranet

Diplomarbeit

Peter Wolter
Studienrichtung: Informatik
Matrikel - Nr. 922230

Fachbereich Informatik
Verteilte Systeme und Betriebssysteme
Johann Wolfgang Goethe - Universität
Frankfurt am Main



betreut durch
Professor Dr. Kurt Geihs

Ehrenwörtliche Erklärung

„Ich versichere hiermit ehrenwörtlich, daß ich die vorliegende Arbeit selbständig und nur unter Benutzung der angegebenen Literatur und Hilfsmittel angefertigt habe. Wörtlich übernommene Sätze und Satzteile sind als Zitate belegt, andere Anlehnungen hinsichtlich Aussage und Umfang unter Quellenangabe kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen."

Frankfurt, den 28. November 1997

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abkürzung	Bedeutung	Seite
ACL	Access Control List	25
ADO	ActiveX Data Objects	70
API	Application Programming Interface	13
ASCII	American Standards Committee for Information Interchange	6
BDC	Backup Domain Controller	46
BIND	Berkeley Internet Name Domain	41
CATID	Category Identifier	87
CCITT	International Telegraph and Telephone Consultative Committee	41
CDS	DCE Cell Directory Service	45
CGI	Common Gateway Interface	15
CLI	Call Level Interface	76
CLSID	Class Identifiers	58
COM	Component Object Model	53
CORBA	Common Object Request Broker Architecture	33
COSS	Common Object Services Specification	36
DAO	Data Access Object	90
DARPA	US Defence Advanced Research Projects Agency	5
DBMS	Database Management System	74
DCE	Distributed Computing Environment	40
DCOM	Distributed Component Object Model	61
DES	Data Encryption Standard	100
DII	Dynamic Invocation Interface	35
DISPID	Dispatch Identifier	57
DLL	Dynamic Link Library	55
DNS	Domain Name Service	45
DSI	Dynamic Skeleton Interface	35
GDS	DCE Global Directory Service	45
DFS	Distributed File Service	45
DTS	Distributed Time Service	45
EPAC	Extended Privilege Attribute Certificates	45
GTZ	Deutsche Gesellschaft für Technische Zusammenarbeit GmbH	1
GUI	Graphical User Interface	6
GUID	Globally Unique Identifier	56
HTML	Hypertext Mark-up Language	6
HTTP	Hypertext Transfer Protocol	6
HTTPS	Hypertext Transfer Protocol with Privacy	16
IANA	Internet Assigned Number Authority	95
IDL	Interface Description Language	34
IID	Interface Identifier	56
IP	Internet Protocol	11
IPC	Inter-process Communication	59
IPID	Interface Pointer Identifier	62
ISAM	Indexed Sequential Access Method	91
ISAPI	Internet Server Application Programming Interface	16
ISO	Intentional Standards Organisation	45
JDBC	Java Database Connectivity	74
JDK	Java Development Kit	18
KDC	Key Distribution Centre	51
LAN	Local Area Network	1

LUW	Local Unit of Work	8
MIDL	Microsoft Interface Definition Language	47
MIME	Multipurpose Mail Extension	15
MIT	Massachusetts Institute of Technology	44
MPNL	Multi Protocol Net Library	95
MS	Microsoft	2
MSRPC	Microsoft Remote Procedure Call	59
NDR	Network Data Representation	59
NFS	Network Files System	13
NISB	Netscape Internet Service Broker	110
NSI	Name-Service Independent	52
NSID	Name Service Interface Daemon	52
OBJREF	Object Reference	62
OCX	OLE Controls	87
ODBC	Open Database Connectivity	16
ODL	Object Description Language	47
ODS	Open Data Services	84
OMA	Object Management Architecture	33
OMG	Object Management Group	33
OOP	Object-oriented Programming	8
ORB	Object Request Broker	33
ORPC	Object Remote Procedure Call	61
OSF	Open Software Foundation	40
OXID	Object Exporter Identifier	62
PSTN	Public Switched Telephone Network	30
PDC	Primary Domain Controller	46
RDO	Remote Data Object	90
RFC	Request for Comments	44
RMI	Remote Method Invocation	17
ROT	Running Object Table	61
RPC	Remote Procedure Call	9
RTT	Round Trip Time	98
SCM	Service Control Manager	62
SDK	Software Development Kit	19
SGML	Standard Generalised Mark-up Language	11
SQL	Structured Query Language	37
SSL	Secure Socket Layer	16
TCP	Transmission Control Protocol	11
TDS	Tabular Data Stream	84
TGT	Ticket Granting Ticket	45
UDP	User Datagram Protocol	11
URL	Uniform Resource Locator	6
VBX	Visual Basic Extensions	87
VM	Virtual Machine	17
VPN	Virtual Private Networks	30
VS	Verteiltes System	7
W3C	World Wide Web Consortium	89
WAN	Wide Area Network	1
Win3x	Microsoft Windows Ver. 3.0 und höher	3
Win95	Microsoft Windows 95	1
WinNT	Microsoft Windows NT Ver. 3.5 und höher	1
WWW	World Wide Web	5

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung und Motivation

1.1 Aufgabenstellung

Im Rahmen der wachsenden Popularität des Internets in der Öffentlichkeit, sehen Firmen in diesem „Wide Area Network (WAN)“ zunehmend auch eine Basis für firmeneigene, verteilte Anwendungen. Zusätzlich werden Technologien, die im Internet erprobt wurden, in das Netzwerk der Firmen übernommen. Diese Architektur bietet ein Modell für dreistufige Anwendungen an, bei dem ein WWW-Browser stets als Front-end dient, der firmeneigene Internetserver inklusive Erweiterungsprogrammen die Applikationslogik repräsentiert und schließlich ein Datenserver als Back-end fungiert. Diese Konfiguration, die sich bereits in klassischen Klient/Server-Umgebungen bewährt hat, verlagert die Anwendungs- und Datenlast weg von den Klienten und hin zu den Servern. Als Hauptvorteil dieser Ausrichtung versprechen Befürworter eine erhebliche Reduktion der hohen Unterhalts- und Supportkosten für Personalcomputer.

Diese dreigeteilte Architektur, bei der „World Wide Web (WWW)“-Browser[1] via „Hypertext Transfer Protocol (HTTP)“ Daten mit WWW-Servern austauschen und bei der WWW-Server ihrerseits über das „Common Gateway Interface (CGI)“ oder proprietäre Schnittstellen[2] mit Applikationen kommunizieren, wird allerdings schon obsolet, bevor sie sich richtig etablieren konnte. Die Nachteile von HTTP als zustandsloses Protokoll, das nach jeder Datenübertragung die Verbindung trennt, und die Beschränkungen des CGI führen zu der Suche nach alternativen Ansätzen, die auf die Lastverschiebung zurück zum Klienten bauen.

Bei Anwendungen, die auf Seite des Klienten ausgeführt werden, hat sich JAVA zweifellos bereits einen festen Platz gesichert. Microsoft versucht dafür seine ActiveX Technologie als Nachfolger von OLE zu etablieren. Neben der zu verwendenden Sprache ist aber auch die Frage nach dem zugrundeliegenden Komponentenmodell zu klären. Gesucht wird eine Infrastruktur, die es erlaubt, über das Netzwerk geladene Anwendungen zu Applikationen zu kombinieren, ohne einen großen Aufwand in die Implementierung der Kommunikation selbst zu investieren. Gleichzeitig soll diese Form der Interaktion die Kommunikation von lokal ausgeführtem Code mit Objekten erlauben, die potentiell weltweit verteilt sind. Die Frage nach einer Lösung kann, in Ermangelung einer Alternative, auf die Entscheidung zwischen der „Common Object Request Broker Architecture (CORBA)“ der „Object Management Group (OMG)“ und dem „Component Object Model (COM)“ der Firma Microsoft reduziert werden.

Für verteilte Objekte sieht Microsoft (MS) „Distributed COM (DCOM)“ vor, welches der Softwarehersteller in einer ersten Implementierung mit „MS Windows NT (WinNT)“ V4.0 ausliefert, und als Zusatz zu „MS Windows 95 (Win95)“ frei verfügbar ist. Weiter fortgeschritten ist die Entwicklung auf Seiten des offenen CORBA-Standards, wobei mehrere Hersteller schon Produkte vorzuweisen haben. Einige von ihnen können mit „Object Request Brokern (ORB)“ aufwarten, die in JAVA geschrieben wurden und damit innerhalb von WWW-Browsern ablaufen können. Sie sind beispielsweise in der Lage JAVA-Applets mit, zu CORBA kompatiblen Objekten, zu verbinden [CW96].

Die „Deutsche Gesellschaft für Technische Zusammenarbeit (GTZ) GmbH“ ist weltweit eines der größten Dienstleistungsunternehmen im Bereich der Entwicklungszusammenarbeit. In 135 Ländern des Südens und Ostens arbeiten mehr als 8.000 Mitarbeiter. Um eine effiziente Kooperation von diesen verteilten Unternehmenseinheiten zu gewährleisten, werden die Außenstellen an das TCP/IP „Local Area Network (LAN)“ der Zentrale in Eschborn über Wahl- und Standleitung mit unterschiedlicher Übertragungskapazität angebunden. Diese Anbindung stellt zur Zeit nur Dienste

[1] grafisches Front-end zur Navigation im WWW.

[2] wie zum Beispiel ISAPI und das Server Plug-in API von Netscape.

für die Telekommunikation zur Verfügung, soll aber in Zukunft für Informationszugriff und -verarbeitung ausgebaut werden. Dabei sind besonders Microsoft ACCESS Datenbanken von Bedeutung. Darüber hinaus werden auch auf INFORMIX und ADABAS/NATURAL Datenbanken Datenbestände von allgemeinem Interesse wie zum Beispiel Telefonlisten, Personaldaten, Adressenverzeichnisse und Informationen zur Kostenverrechnung bereitgestellt. Zunehmend nimmt auch der SQL Server der Firma Microsoft an Bedeutung zu.

In der vorliegenden Arbeit sollen sowohl Sprachen als auch die verfügbaren Komponentenmodelle verglichen werden. Exemplarisch wird ein „Active View“ auf ODBC-Datenquellen in einer 3-Schicht Architektur implementiert, um so Defizite der Technologien im praktischen Einsatz aufdecken zu können. Besonderes Augenmerk wird dabei auf die technischen Voraussetzungen der Infrastruktur des Auftraggebers der Arbeit gelegt. Weiter werden die unterschiedlichen Anforderungen durch die Verteilung von Objekten in Internet- und Intranetarchitekturen untersucht, eine Differenzierung vorgenommen und die Leistungsfähigkeit der Systeme auf den engbandigen Übertragungskanälen des Internets eruiert.

Die exemplarisch zu erstellende Klienten-/Serveranwendung, auf Basis von verteilten Objekten, soll in einem verteilten, offenem System Informationszugriff und -verarbeitung erlauben, um so die Möglichkeiten des Einsatzes im allgemeinen und für die GTZ im besonderen bewerten zu können. Eine solche Aufgabenstellung wird auch eine analytische Betrachtung von verteilten Objekten auf TCP/IP basierten Netzen, unabhängig von deren Einsatz in Datenbankanwendungen ermöglichen. Im Rahmen dieser Betrachtung stehen vor allem Verfügbarkeit, Leistungsfähigkeit und Sicherheit der Modelle im Vordergrund.

Diese zu realisierende Anwendung besteht grundsätzlich aus zwei Komponenten:

1. Eine Serveranwendung greift über eine Schnittstelle auf beliebige Datenbanken zu und stellt die enthaltenen Informationen über eine definierte Schnittstelle zur Verfügung. Hierbei soll die Struktur der Datenbank nicht vorgegeben, sondern von der Anwendung erkannt und weiterverarbeitet werden können. Auf diesem Weg sollen Datenbanken auf verschiedenen Systemen als unternehmensweite Datenbasis zur Verfügung gestellt werden.

2. Die Klientenanwendung soll dem Anwender sowohl im LAN als auch im WAN eine ergonomische und von der Datenstruktur unabhängige Oberfläche zur Informationsverarbeitung zur Verfügung stellen. Dabei soll ein „Benchmark“-Vergleich die Möglichkeit der Objektimplementierung auf dem WWW unter JAVA und der Entwicklung einer verteilten ActiveX-Komponente unter MS Visual Basic gegenüberstellen. Besonders die Einwirkungen von unterschiedlichen Übertragungszeiten sind zu untersuchen und mit denen in klassischen LANs zu vergleichen.

Folgende Aspekte, die zumeist aus der Verteilung der Objekte und der variablen Bandbreite der Netzwerke resultieren, treten in diesem Zusammenhang bei der Entwicklung auf und müssen als Vorbereitung betrachtet werden:

1. Sicherheitsaspekte eines Intranet und des Internets:
   Hierbei soll besonders die Authentifizierung der Anwender, als auch die Sicherheit der Übertragung selbst diskutiert und eine geeignete Lösung mit Rücksicht auf geringe Bandbreiten erarbeitet werden. Die Anforderungen an die Sicherheit in einem Intranet und dem Internet werden gegenübergestellt und die beiden Topologien gegeneinander abgegrenzt.

2. Besonders das Transaktionsmanagement bei variablen Übertragungszeiten muß realisiert werden. Hier soll die Möglichkeit eines dynamischen Ansatzes verfolgt und durchgeführt werden.

3. Der Erstellung einer ergonomischen, aber dennoch flexiblen Oberfläche kommt eine besondere Bedeutung zu, da sich die Anwender in den Außenstellen der GTZ erst seit kurzem mit der Problematik der EDV befassen meist kaum nachträglich geschult werden können.

4. Die Möglichkeit der Zugriffsbeschleunigung durch „Prefetching“ - dem frühzeitigen Abfragen von Daten - soll geprüft werden.

Um eine umfassende Bearbeitung der Aufgabenstellung durchzuführen, wird abschließend ein Ausblick auf zu erwartende Technologien in diesem Bereich erfolgen. Die Middlewaremodelle werden ausführlich verglichen und die Möglichkeit des Einsatzes im Internet und Intranet evaluiert. Dadurch wird auch dem Einsteiger in diese Materie eine fachlich fundierte Auswahl eines der Systeme ermöglicht.

1.2 Einschränkungen

Neben den allgemeinen Betrachtungen in der vorliegenden Arbeit unterliegen die praktischen Ausführungen Einschränkungen, die sich durch die Soft- und Hardware Infrastruktur der GTZ ergeben. Diese wird in der folgenden Abbildung schematisch verdeutlicht.

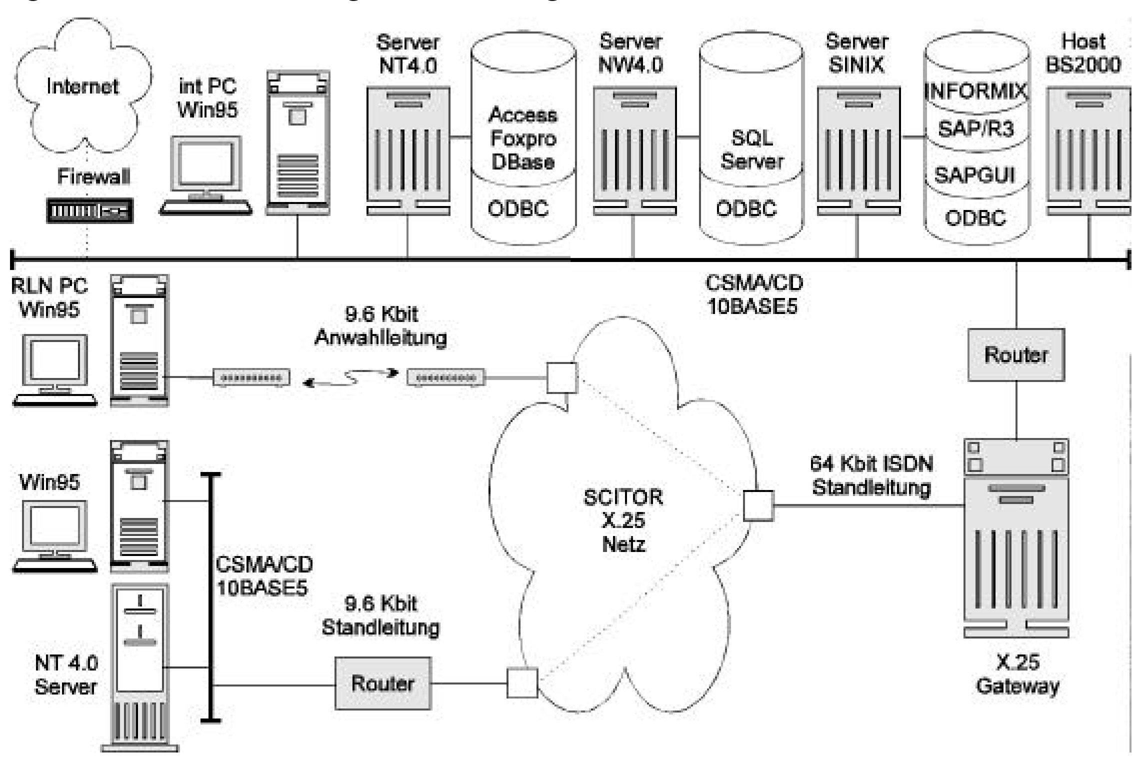


Abb. 1: Hard- und Software Infrastruktur der GTZ.

Das Intranet der GTZ besteht aus zwei Regionen, die sich in der maximalen Übertragungsgeschwindigkeit unterscheiden. In der Diskussion über verteilte Objekte im Intranet ist die 9.6KBit Verbindung der internationalen Außenstellen besonders zu beachten. Das Verhalten der Middleware-Technologien (COM und CORBA) auf langsamen Verbindungen muß also separat betrachtet werden, gerade weil eine Aufrüstung auf schnellere Kanäle aus Kostenüberlegungen in absehbarer Zeit nicht geplant ist. Die Zieltechnologie muß in beiden Regionen performant sein, um Akzeptanz bei dem Auftraggeber zu finden.

Da der kommerzielle Netzwerkanbieter SITA die Sicherheit seines SCITOR X.25 Netzes garantiert, können diese Außenstellen der GTZ aus Sicherheitsüberlegungen ohne Einschränkungen zum Intranet gezählt werden. Die Aspekte der Sicherheit von Middleware müssen also vor allem jenseits der sogenannten Firewall, und damit im Internet betrachtet werden. Die Eignung für den Einsatz im Internet hängt vor allem von Performanzüberlegungen im Lichte hoher Anwenderzahlen und Aspekten der Softwareverteilung ab.

Seitens der eingesetzten Software werden auf Seite der Klienten ausschließlich Betriebssysteme der Firma Microsoft, also Win3x, Win95 und WinNT eingesetzt. In der Beurteilung der

praktischen Anwendbarkeit von Middlewaretechnologien kann hier ein homogenes Softwaresystem angenommen werden, obwohl der Einsatz auf anderen Systemen nicht aus den Augen verloren werden darf. Die Heterogenität der eingesetzten Datenbanken grenzt die praktische Betrachtung auf eine standardisierte Datenbankschnittstelle ein, die alle eingesetzten Datenbanken unterstützt. Hier kommt die noch vorzustellende JDBC-Schnittstelle nicht direkt in Frage, da Treiber zur Zeit noch nicht von allen Datenbankherstellern zur Verfügung gestellt werden. Als einzige gangbare Alternative, die sowohl von JAVA als auch von ActiveX unterstützt wird, wird in dieser Arbeit auf die weitverbreitete und standardisierte ODBC-Schnittstelle aufgesetzt.

2 Internet und Intranet

2.1 Definitionen

Seine Wurzeln findet „das Internet” in einem öffentlichen und heterogenen WAN, das 1969 in den USA entwickelt wurde. Dieses frühe DARPA Internet - ARPANET - wurde zur Verbindung von Computernetzen der amerikanischen Universitäten mit denen der DARPA errichtet. Der Name stammt von der fördernden Abteilung „Advanced Research Project Agency“ der „US Defence Advanced Research Projects Agency (DARPA)“ des Verteidigungsministeriums der USA. Zunächst diente es lange Zeit experimentellen und theoretischen Untersuchungen auf dem Gebiet der Rechnernetze. Im Laufe der Zeit verband es eine Vielzahl von Universitäten und Forschungseinrichtungen und konnte auch von anderen Ländern und Erdteilen erreicht werden. Seit der Gründungszeit wächst dieser Verbund von Netzwerken ständig. Anstelle von wenigen Mainframe Computern findet man heute eine Vielzahl von Workstations. Weiter bedeutete die Einführung von LANs das Entstehen von einigen Tausend Teilnetzen. Das heutige Internet, als Nachfolger des ARPANET, ist mit einer Vielzahl von anderen Netzwerken verbunden. Dieser Verbund von Netzwerken wird in der Öffentlichkeit (etwas ungenau) als *das Internet* bezeichnet [HAL96].

Schätzungsweise 50 Millionen Menschen sind mit dem Internet vertraut, können elektronische Nachrichten (E-Mails) versenden, Informationen suchen und finden und unterschiedlichste Dienste in Anspruch nehmen. Für diese weite Verbreitung ist die Einführung des „World Wide Web (WWW)“ als grafische Benutzeroberfläche („graphical user Interface (GUI)“) verantwortlich, mit welcher der eigentlichen Durchbruch des Internets in der Öffentlichkeit gelungen ist. Mittels diesem ergonomischen und intuitiv zu bedienenden Dienstes ist es möglich, Informationen jederzeit und an jedem Ort zu recherchieren. Der Gedanke liegt nun nahe, die im Internet erprobten Techniken innerhalb eines Unternehmens einzusetzen [VIN96] und als Plattform für verteilte Anwendungen auf Objektbasis zu verwenden.

Es wird deutlich, daß diese semantische Interpretation des Begriffes Internet keine Abgrenzung zu dem des Intranets zuläßt. Um eine solche Trennung definieren zu können, muß auf den entscheidenden Unterschied des Internets zu herkömmlichen LANs zurückgegriffen werden. Es beruht auf offenen Standards, zu denen primär das Netzwerkprotokoll TCP/IP gehört. Für eine detaillierte Übersicht über die „Internet Protocol (IP)“-Standards siehe [HAL96] und [TAN97]. Übernimmt man TCP/IP aus dem Internet in das eigene lokale Netzwerk und installiert zugehörige Dienstserver (z.B. WWW, GOPHER, FTP, Telnet, etc.) gewinnt das Unternehmen automatisch „Data-sharing“ und „Information-Retrieval“ Kapazitäten. Zunächst ergeben diese Überlegungen eine Definition für den Begriff des Internets [BRU96]:

> *Das **Internet** ist ein öffentliches WAN auf Basis des TCP/IP.*

Die Übernahme dieser Technologien in lokale Netze tangiert allerdings immer das Problem der Daten- und Netzwerksicherheit. Um das Unternehmensnetzwerk vor dem Zugriff Außenstehender zu schützen werden sogenannte Firewalls zur Trennung der Netzwerke verwendet. Die Funktionsweise, Bedeutung und Definition von Firewalls werden im Kapitel 2.4.3.2 näher erläutert. Schon jetzt kann mit Hilfe dieser Überlegung der Begriff selbst definiert werden:

> *Ein **Intranet** ist entweder ein, durch eine Firewall vom Internet getrenntes, Teilnetz oder ein völlig autarkes Computernetzwerk auf Basis des TCP/IP ohne Verbindung zum Internet.*

Der Unterschied zwischen den beiden Begriffen liegt also lediglich im Sicherheitsaspekt. Solange dieser Punkt nicht für eine Diskussion in Betracht gezogen werden muß, sollen die Begriffe Internet und Intranet synonym gebraucht werden. Bei einer Trennung der Begriffe wird gesondert auf den Sachverhalt hingewiesen. Beachtenswert ist weiter, daß der Begriff Internet weder eine

Aussage über die Topologie noch über die Ausdehnung eines Computernetzes trifft. So dehnt sich zum Beispiel das Intranet der GTZ als WAN weltweit aus. Inhaltlich vom Begriff des Internets ist der des WWW klar zu trennen. Das WWW ist eine Anwendungsumgebung auf Basis des Klient/Server-Modells, die sich aus mehreren Komponenten zusammensetzt:

Das „Hypertext Transfer Protokoll (HTTP)“ definiert die Kommunikation zwischen dem WWW-Browser (Klient) und dem WWW-Server. Dieses verbindungslose Protokoll definiert den Austausch von HTML Dokumenten. Hierbei übernimmt der Klient eine aktive Rolle: sobald ein Dokument angefordert wird, baut der Klient eine Verbindung zum Server auf, der Server sendet das Dokument und die Verbindung wird anschließend abgebaut [BRU96]. Es ist dabei nicht möglich, vom Server aus Zustandsinformationen über den Klienten zu erlangen. Die „Hypertext Mark-up Language (HTML)“ ist eine Sprache um „American Standards Committee for Information Interchange (ASCII)“ [HAL96] Dokumente mit Strukturierungsanweisungen zu versehen, damit Informationen plattformunabhängig dargestellt werden können. Dokumente werden dabei mit sogenannten *Hyperlinks* verbunden. Diese Hyperlinks ermöglichen den einfachen Übergang zu neuen Informationsquellen im HTML Standard, die durch einen „Uniform Resource Locator (URL)“ eindeutig identifiziert werden. URLs identifizieren HTML-Seiten dergestalt, daß Fragen nach dem Namen der Seite, deren Lokation und der Methode, wie auf die Seite zugegriffen werden kann, durch Zuweisung eines einzigen, global eindeutigen Bezeichners gemeinsam beantwortet werden. URLs bestehen demnach aus drei Teilen: dem Protokoll, dem DNS-Namen der Maschine (vgl. Kapitel 3.2.2.5) und einem lokal eindeutigen Namen der spezifischen Seite (üblicherweise der Dateiname) [TAN97].

Das *WWW* besteht also aus Software, Protokollen, Konventionen und Informationen, welche die Veröffentlichung von Hypertext Dokumenten und Multimedia Ressourcen auf verschiedenen Computern weltweit ermöglichen. Damit bildet es ein „Netz“ von Informationsquellen.

2.2 Systemmodelle

2.2.1 Das Grundmodell für zentrale Systeme

In der Diskussion um verteilte Objekte müssen zunächst die grundlegenden Begriffe geklärt werden. In der Informatik versteht man unter einem *System* die Zusammenfassung mehrerer Komponenten zu einer als ganzes aufzufassenden Einheit (vgl. [HER93] für Erkennungsmerkmale von Systemen und weitere Unterscheidungen). Die meisten Anwendungen, oder Anwendungssysteme, bestehen aus drei Grundelementen: einer, heute meist grafischen, Benutzerschnittstelle (GUI), einer berechnenden Einheit und einem Informationsspeicher. Im allgemeinen Rechnermodell sind die einzelnen Komponenten integriert, so daß eine Differenzierung kaum möglich ist. Aber auch in diesem Fall sind gute Entwurfsprinzipien wie Modularität, genaue Schnittstellenspezifikation und universelle Verwendbarkeit wichtig, um eine spätere Modifikation an der Anwendung leichter durchführen zu können und um die Fehlersuche zu vereinfachen. Weiter führt die klare Definition der Komponentenschnittstellen zur Wiederverwendbarkeit der Bausteine und so zu schnellerer Amortisierung und besserer Sicherung des Entwicklungsaufwandes.

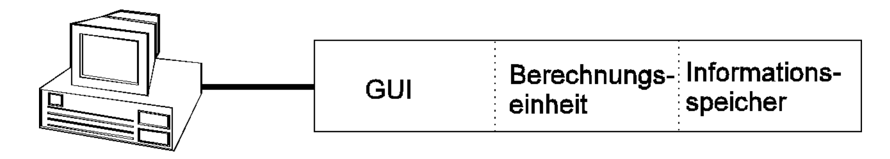


Abb. 2: Ein zentrales Systemmodell [ROS93].

2.2.2 Verteilte Systeme

Die einfachste Möglichkeit ein System zu verteilen ist das System an den Schnittstellen seiner Komponenten aufzutrennen und damit den Aufgaben gemäß zu partitionieren. So kann jede Komponente in der für sie optimalen Umgebung entwickelt und auch in einer solchen ausgeführt werden. Zum Beispiel können starke Inanspruchnahme durch Anwender oder hoher Rechenaufwand innerhalb einer Komponente zur der Entscheidung führen, eine bestimmte Komponente auf einem dedizierten System zu aktivieren. Durch Verteilung können also „Flaschenhälse“ im System vermieden oder reduziert werden.

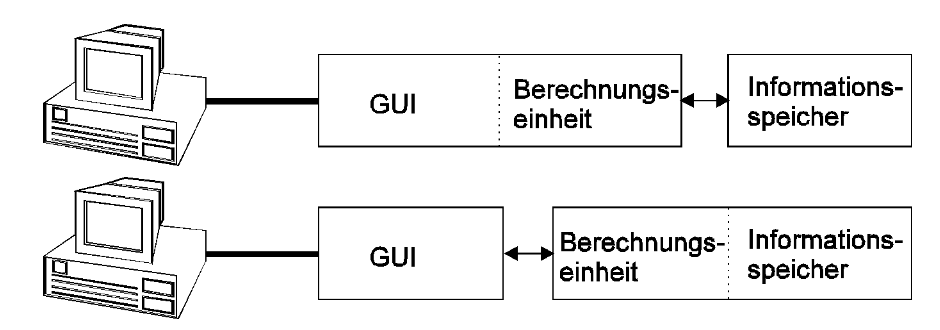


Abb. 3: Ein einfaches verteiltes System [ROS93].

> *Ein **System** heißt **verteilt**, wenn sich seine Komponenten an räumlich getrennten Stellen befinden (können), hierdurch aber die Funktionalität des Gesamtsystems nicht beeinträchtigt wird* [HER93].

Solche „verteilten Systeme (VS)“ sind gegenüber „monolithischen Systemen“ stabiler gegen Ausfälle, leichter zu skalieren und flexibler, fordern aber Verantwortung und Kompetenz am Arbeitsplatz, da sie schwerer zu administrieren sind. Für die Informatik stellen VS vielfältige Herausforderungen in Bereichen der parallelen Verarbeitung, Beschreibung, Synchronisation, Sicherung und Gewährleistung der Zuverlässigkeit dar [HER93]. Unter Umständen kann es auch sinnvoll sein, eine Komponente selbst auf verschiedene Systeme zu verteilen. Dieser Ansatz ermöglicht eine besondere Form des Zusammenwirkens zwischen den einzelnen Teilsystemen, die in zentralen, also nicht verteilten Umgebungen, nicht erreichbar wäre.

Es ist klarzustellen, daß die Verteilung eines Systems immer einen erhöhten Kommunikationsaufwand zwischen den Komponenten nach sich zieht, der in einem zentralen System so nicht auftreten würde. Die strategische Entscheidung für oder gegen den Einsatz eines VS kann also nur gekoppelt an eine Überlegung zur Kommunikationsstruktur in diesem System erfolgen. Das Kommunikationssystem kann schnell zum Flaschenhals werden und damit Vorteile der Verteilung egalisieren.

2.2.3 Das Klient/Server-Modell

Verteilte Systeme folgen zumeist dem Klient/Server-Modell, dessen Definition stark von dem jeweils adressierten Problem abhängig ist. Die vorliegende Arbeit wird folgende Definition verfolgen:

> *Ein **Klient** ist ein Prozeß, der einen Dienst eines Servers anfordert. Ein **Server** ist ein Prozeß, der entsprechende Anforderungen des Klienten nach einem Dienst beantwortet. Das **Klient/Server-Modell** beschreibt den Ansatz, Rechensysteme in Kliententeile und Serverteile zu segmentieren.*

Die Mehrheit der offenen, verteilten Anwendungen basieren auf diesem Klient/Server-Modell. Zum Beispiel verwendet ein Anwendungsprogramm (Prozeß) ein entferntes Dateisystem (Anwendungsprozeß). Das Dateisystem ist damit der Server, da es ausschließlich auf Dienstanforderungen an das Dateisystems reagiert, und das Anwendungsprogramm ist der Klient, da es

ständig solche Anforderungen generiert [HAL96]. Es ist zu beachten, daß ein einzelner Serverprozeß normalerweise mehrere Anforderungen von einer verteilten Gemeinschaft von Klienten gleichzeitig beantworten muß.

Ein wichtiges Konzept in verteilten Systemen ist daher die logische *Arbeitseinheit*, die „local unit of work (LUW)“. Diese LUW ist eine Menge von Prozessen, die koordiniert und zu einem Ergebnis hin ausgeführt werden müssen. Auch nur ein Fehler in einem einzelnen (Teil-) Prozeß führt dazu, daß das gesamte VS auf einen Zustand vor der Ausführung des LUW zurückgesetzt wird [BRU97].

2.2.4 Objekte und Objektorientierung

Bei jeder Art der Entwicklung von Software ist der Ausgangspunkt eine mehr oder weniger präzise beschriebene Aufgabenstellung. Bei der Überführung dieser Beschreibung in ein Softwaresystem müssen die so spezifizierten inhaltlichen Anforderungen vollständig erfüllt werden. Der *prozedurale* Ansatz geht davon aus, daß ein Programm aus Daten besteht, welche von Prozeduren bearbeitet werden. Aus der *deklarativen* Perspektive besteht ein Programm aus einer Menge von Regeln, die auf Fakten angewendet werden. Beim Grundgedanken der *Objektorientierung* wird ein Programm als eine Menge eigenständiger Objekte betrachtet, die der realen Welt nachgebildet werden.

In dieser Arbeit findet grundsätzlich die aus der objektorientierten Programmierung („Object-Oriented Programming (OOP)“) [STR92] bekannte Definition von Objekten Anwendung:

> *Ein **Objekt** ist ein Informationsträger, der einen (zeitlich veränderbaren) Zustand besitzt und für den definiert ist, wie er auf eingehende Nachrichten reagiert; die Definitionen der Nachrichten können sich im zeitlichen Ablauf verändern, so daß ein Objekt auf eine Nachricht zu unterschiedlichen Zeitpunkten unterschiedlich reagieren kann* [HER93].

Die objektorientierte Programmierung erfolgt also durch Senden von Nachrichten an Objekte, wodurch Objekte in ihrem Zustand verändert werden oder Informationen über ihren Zustand erhalten werden können. Ein Objekt der realen Welt legt eine Vielzahl von Verhaltensweisen an den Tag. Diese werden als Methoden realisiert und die Charakteristiken werden zu seinen Variablen.

Der objektorientierte Ansatz zur Erstellung von Software Systemen hat weit mehr Vorteile als nur Flexibilität bezüglich der Aufgabenerfüllung. Weil die Struktur der Software die reale Welt reflektiert, kann diese leichter verstanden und nachvollzogen werden. Das bedeutet, daß eine auf unternehmensweiten Modellen basierende Software eine weit längere Lebensdauer hat als Programme, die nur zur Lösung von spezifischen, unmittelbar anstehenden Problemen geschrieben werden. Objekte und deren Kombination können, im Unterschied zum klassischen Programmiermodell, verhältnismäßig leicht neuen Vorgaben der Firmenstruktur oder neuen Prozeßabläufen nachgebildet werden. Im Gegensatz zur prozeduralen Programmierung sind Objekte allgemeingültige Bausteine, die dadurch leicht unter anderen Aufgabenstellungen wiederverwendet werden können. Die Ausrichtung des Entwicklungsprozesses auf die Erstellung allgemein verwendbarer Modelle hat damit zur Folge, daß diese sich viel einfacher an neue Anforderungen anpassen lassen. Schließlich führt die umfassende Wiederverwendung von bereits existierenden, geprüften Komponenten nicht nur zu kürzeren Entwicklungszeiten, sondern auch zu viel robusteren, fehlerfreien Systemen.

Der konsequente Einsatz objektorientierter Techniken in allen Phasen des Software Lebenszyklus verkürzt also die Entwicklungszeit der Systeme, erhöht deren Qualität und Flexibilität und erleichtert die Wartung. Eine auf Objekten basierende Denkweise ermöglicht ein durchgängiges Konzept von der Analyse bis zur Implementierung. Neben der besseren Verständlichkeit sind gute Erweiterbarkeit, Wiederverwendbarkeit und einfache Wartung weitere Vorteile der objekt-

orientierten Technologie, die zur Kostensenkung beitragen. Der Einsatz des Paradigmas der Objektorientierung ist also im allgemeinen anzustreben.

Die allgemeine Definition von Objekten wird in den folgenden Kapiteln bei Bedarf ergänzt und erweitert. Auf Unterschiede zu dieser grundsätzlichen Definition wird gesondert hingewiesen.

2.2.5 Middleware

Middleware ist eine Sammlung von Softwarediensten, welche die Implementierung von Klient/Server-Modellen unterstützen. Sie kann zum Beispiel eine gemeinsame Schnittstelle und eine Auswahl von Diensten anbieten, wenn verschiedene Systemplattformen Anwendung finden. Fortgeschrittene Middleware kann weiter die Lokation der Dienste und anderer Anwendungen vor dem Benutzer verbergen.

> *„**Middleware** ist eine Softwareschicht, welche eine gemeinsame Schnittstelle und Übersetzung zwischen den Anwendungen, den Daten und dem Betriebssystem bietet* [BRU97].*“*

Es gibt verschiedene *Dienste* die Middleware anbieten kann. Die folgende exemplarische Aufzählung der wichtigsten Dienste ist nicht ausschließlich:

- Verzeichnis: Verwaltet Standort und Eigenschaft von Netzwerkressourcen.
- Sicherheit: Sicherheitsmechanismen sowohl für Benutzer und schutzbedürftige Netzwerkressourcen.
- Management: Administrative Aufgaben der Middleware.
- API: Eine konsistente Schnittstelle für Anwendungs- und Dienstimplementierung.
- Zeit: Einheitliches Zeitsystem um Dienste zeitlich koordinieren zu können.

Es gibt viele Ansätze für Middleware um die Interaktion zwischen Klienten und Server zu unterstützen. Grundsätzlich fallen diese in drei verschiedene *Middlewaremodelle* [BRU97]:

Das *Conversational Model* bietet Dienste für eine einfache Kommunikation. Der Klient befragt den Server und wartet dann auf eine Antwort. In einigen Definitionen von Netzwerkprotokollen wird dieses Verfahren auch als v*erbindungsorientiert* bezeichnet. Dies impliziert, daß eine Verbindung zwischen Klient und Server bestehen muß, damit Kommunikation stattfinden kann.

Das *„Remote Procedure Call (RPC)“ Model* bietet den Mechanismus des Prozeduraufrufes auch für entfernte Server an. Ein *Prozeduraufruf* ist eine Anweisung, welche die Ausführung des Rumpfes einer vereinbarten Prozedur bewirkt. Bei *Funktionsprozeduren* wird nach Verarbeitung des Rumpfes ein Ergebniswert zurückgegeben [HER93]. Dieses Modell ist synchron, da es von der koordinierten Ausführung von Klient und Server abhängig ist.

Das *Messaging Model* verwendet Nachrichtenschlangen („Queues“), um die Anforderungen zwischen Klient und Server zu übertragen. Eine Folge heißt *Schlange*, wenn Elemente nur am Ende eingeführt und am Anfang entfernt werden können. Jeder Prozeß observiert seine Queue auf eingegangene Nachrichten, um diese dann zu verarbeiten. Dieses Modell erlaubt die Implementierung von nachrichtenorientierter anstelle von transaktionsorientierter Verarbeitung. Damit kann es als asynchron angesehen werden, da die Ausführung von Klienten und Server unabhängig voneinander erfolgt. In einigen Definitionen von Netzwerkprotokollen wird dieses Modell auch als *verbindungslos* bezeichnet.

Die einzelnen Middleware Modelle sind in der folgenden Abbildung zusammengefaßt:

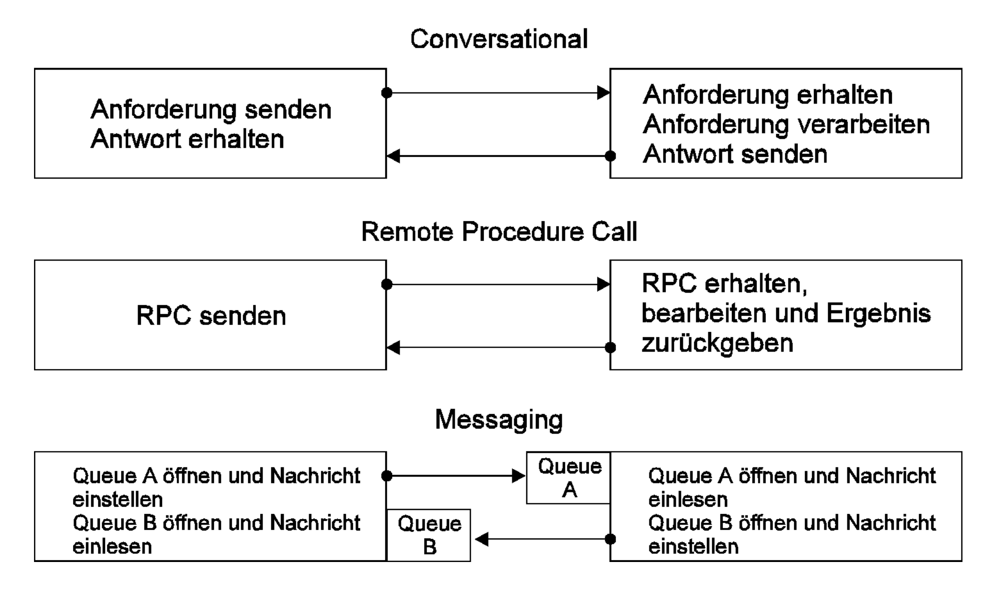


Abb. 4: Middlewaremodelle [BRU97].

Die Middleware-Technologie ist ein großer Schritt in Richtung modularer und flexibler Anwendungen, die über einen Softwarebus oder Kommunikationsmechanismus miteinander kommunizieren. So wandelt sich das klassische Klienten/Server-Modell zu einem Modell, bei dem jeder Computer Klient und Server zugleich sein kann. Auf jedem Computer arbeiten verteilte Komponenten, die auch von anderen Systemen benutzt werden. Sie stellen selbstverwaltende, „intelligente“ und autarke Einheiten dar, die sich ändernden Anforderungen angepaßt werden können. Zudem ist dieses gesamte System offen für Anpassungen und Erweiterungen. Module (einzelne verteilte Komponenten) können durch andere ersetzt werden, ohne die übrigen Module zu beeinflussen. Die Überlegungen der vergangenen Abschnitte führen abschließend zu der Integration von OOP und Middleware. Der Ansatz, Objekte über eine solche Middleware auf ein System zu verteilen, führt zu der in dieser Arbeit verwendeten Definition von verteilten Objekten:

> ***Verteilte Objekte*** *sind Objekte im Sinne der objektorientierten Programmierung, die prozeßübergreifend über eine Middleware kooperieren.*

Mehrere verteilte Objekte können damit ein VS oder eine LUW bilden. Weitere Details zum Thema Middleware wird in den Kapiteln zur Verarbeitung mit verteilten Objekten behandelt.

2.3 Aktuelle Technologien verteilter Anwendungen im Internet

2.3.1 Grundlagen aktueller Technologien

In den 50er und 60er Jahren wurden im Rahmen der Entwicklung von Großrechnern (~ Mainframes) fast nur zentrale Systeme eingesetzt, bei denen die Kontrolle und Verarbeitung ausschließlich bei dem zentralen Rechensystem lag. Aus Gründen der Effizienz mußten sich Unternehmen immer mehr den Anforderungen der EDV unterordnen. Gleichzeitig erforderten zentrale Programme immer größere Computer. Nicht zuletzt aus betriebswirtschaftlichen Betrachtungen heraus wurden so langsamere Mehrplatzsysteme oder Netzwerke direkt bei den einzelnen Anwendern installiert, die in ihrer Summe sogar die Rechenleistung des Großrechners übertrafen.

Grundsätzlich sind VS von ihrem Netzwerk unabhängig zu betrachten, da dieses lediglich als Basis für die Kommunikation der einzelnen Komponenten des VS dient. Ein solche Grundlage bietet zum Beispiel das Internet auf Basis des TCP/IP Protokolls, welches in dieser Arbeit Verwendung findet. Dieses Protokoll entstand im Rahmen der Forschung am ARPANET. Das

„Internet Protocol (IP)“ arbeitet auf der dritten Schicht des OSI-Modells [TAN97] mittels Datagrammen, welche in jedem Paket eine vollständige Adressierung sowohl des Empfängers als auch des Absenders beinhalten. Für eine Übersicht der Protokollschichten des IP-Modells siehe auch [HAL96]. Dadurch entfallen komplexe Verbindungsoperationen (wie zum Beispiel bei X.25), allerdings wächst die Größe der Datenpakete aufgrund der Paketköpfe stark an. Das „Transmission Control Protocol (TCP)“ wird auf der vierten Schicht zur verbindungsorientierten Kommunikation im Internet verwendet und benutzt dazu den verbindungslosen IP-Dienst. Durch diese Kombination wird eine gesicherte Verbindung angeboten. Als Alternative bietet das „User Datagram Protocol (UDP)“ ein „unsicheres“ und verbindungsloses Datagrammprotokoll an [HER93].

Eine *IP-Adresse* ist eine 32bit Nummer, die innerhalb einer administrierten Domäne (z.B. eines Intranet) eindeutig ist. Eine Netzwerkkomponente kann dabei, für jeden ihrer Netzwerkadapter, jeweils eine IP-Adresse zugewiesen bekommen. Ein *Port* ist der Dienstübergang einer Anwendung auf einem Computer, die durch eine 16bit Nummer identifiziert wird. Im Gegensatz zur IP-Adresse ist die Port Nummer nur innerhalb der Netzwerkkomponente eindeutig. Damit identifiziert die Kombination von IP-Adresse und Port eindeutig einen Dienst einer Netzwerkkomponente.

Das Hauptargument für den Einsatz des Internets in Unternehmen (Intranet) ist eine einheitliche Plattform, die weltweite Kompatibilität garantiert, und zudem äußerst flexibel den unternehmensweiten Arbeitsabläufen angepaßt werden kann. Hinzu kommt die ausgezeichnete Eignung als Basis für verteilte Anwendungen. Betrachtet man ein Netzwerk auf Basis von TCP/IP, sei es nun eine Intranet- oder Internettopologie, gibt es verschiedene aktuelle Middleware-Technologien. Zum einen wurden die bekannten Technologien für TCP/IP basierte LANs auf das Internet portiert, zum anderen sehen wir neue Entwicklungen von „echten“ Middlewareplattformen für das Intranet.

2.3.2 Hypertext Mark-up Language (HTML)

Die breite Palette von Computern, Darstellungsformaten und Softwareplattformen bei heterogenen Netzwerken erfordert eine Strategie, in welcher der Klient die Hauptrolle in der Entscheidung spielt, wie erhaltene Informationen verarbeitet und/oder angezeigt werden. Eine Steuerung der Verarbeitung seitens des Servers ist nicht praktikabel. So wurde eine Sammlung von Internetprotokollen erstellt, die Adressierung, Präsentationsformate und Verarbeitung fremder Formate berücksichtigt, um so Interoperabilität zu ermöglichen.

HTML ist die Basissprache, die von allen WWW-Browsern verstanden wird. Nicht modifiziertes und zum Standard konformes HTML ist damit plattformunabhängig und sehr einfach in der Implementierung. Diese Sprache wurde als Teil des WWW bei CERN von Tim Berners-Lee entwickelt. Sie entstand aus der „Standard Generalised Mark-up Language (SGML)“, der ISO Standardsprache für Text. Dabei führt HTML die Prinzipien von SGML für das Internet fort, indem eine Untermenge der Semantik von SGML mit ähnlicher Syntax implementiert wird [HAL96].

HTML ist eine „Mark-up Sprache“, nicht eine vollständige Programmiersprache. Ein HTML Dokument (Programm) ist ein ASCII Text mit eingefügten Anweisungen (Mark-ups), welche die Darstellung des Textinhaltes beeinflussen. Das Grundmodell für die Ausführung von HTML besteht aus der Anforderung eines Dokumentes mittels des Namens (z.B. URL), Interpretation der enthaltenen HTML Sprache und Darstellung der Information. Das HTML Äquivalent einer Anwendungen besteht somit aus einer Sammlung von verwandten WWW-Seiten die durch einen einzigen HTTP Server (HTTP ist das TCP/IP Protokoll, das für die Interaktion von WWW-Browsern und Servern verwendet wird. Vergleiche dazu auch Kapitel 2.3.4) veröffentlicht werden. Dies ist zwar eine starke Vereinfachung des Modells, entspricht aber der Einfachheit der Sprache selbst und stellt damit gleichzeitig einen ihrer größten Vorzüge dar. Im Rahmen der

Weiterentwicklung von HTML wird die Sprache zwar von einigen Herstellern (z.B. Mircosoft und Netscape) proprietär erweitert, und verliert dabei Teile ihrer Einfachheit, bleibt aber eine brauchbare und praktikable Sprache. Defizite stellt ein Entwickler zumeist nur in zwei Gebieten fest: die Performanz in bestimmten Anwendungstypen und die Fähigkeit bestimmte, häufig verwendete Aufgaben zu programmieren.

HTML ist in seiner Mächtigkeit eingeschränkt. Dies ist ein Entwurfsziel der Sprache, welches die Ausführung von gefährlichen Anwendungen auf Klientenmaschinen verhindert. Leider werden die Möglichkeiten von Internetanwendungen auf HTML-Basis damit auch stark limitiert. Aufgaben, die nicht in HTML implementiert werden können, müssen in einer anderen Sprache realisiert und auf dem Server ausgeführt, oder von dem Server geladen und auf dem Klientensystem ausgeführt werden. Diese erzwungene Segmentierung ist in den meisten Fällen nicht optimal bezüglich Performanz und Modellintegrität und führt ihrerseits wieder zu einer Sicherheitslücke.

Die eingeschränkte Funktionalität von HTML, und der sich daraus ergebenden Verlagerung von Rechenleistung zum Server, hat negative Auswirkungen auf die Performanz der Sprache. Einerseits hat die hohe resultierende Kommunikation der Komponenten, besonders auf langsamen Internetverbindungen, eine negative Auswirkung auf die unmittelbare Antwortzeit, anderseits führt die Umlegung der Rechenlast auf den Server zu einer Erhöhung der Serverlast selbst. Vorhandene Rechenleistung auf der Klientenseite bleibt unbenutzt. Durch die Umstellung in Unternehmen von Einzelplatzsystemen auf vernetze Systeme ist diese Ressource auch als wirtschaftlicher Faktor nicht zu vernachlässigen. Diese Überlegung führt zu der folgenden begrifflichen Differenzierung und Diskussion um die Distribution von Programmen im allgemeinen, die ebenfalls Grundlage für die Erörterung von verteilten Objekten ist:

Wenn Anwendungen auf einem Klientensystem ausgeführt werden sollen, gibt es zwei Hauptaspekte: was wird verschickt und was wird ausgeführt. Dabei gibt es jeweils drei Alternativen: Quellcode, ein teilweise kompiliertes Zwischenformat (z.B. Bytecode) und Binärcode. Da die Kompilation auf dem Klienten vorgenommen werden kann, stimmen übertragene und ausgeführte Information nicht zwingend überein.

Bytecode (nach Messungen die auf der JavaOne Konferenz vorgestellt wurden) kann 2-3 mal kleiner als vergleichbarer *Binärcode* sein. Damit ist der Transfer der Information erheblich schneller, was sich besonders auf langsamen Verbindungen bemerkbar macht. *Quellcode*, wie er auch bei HTML Verwendung findet, ist ebenfalls sehr kompakt. Bezüglich Ausführungsperformanz ist Binärcode der Verwendung von Bytecode vorzuziehen, der die Performanz von Quellcode wiederum übertrifft. Generell kann Binärcode 10-100 mal schneller ausgeführt werden als Bytecode. Viele JAVA VM (Virtual Machine) Entwickler arbeiten zum Beispiel an der Entwicklung von JIT (Just in Time) Compilern, um die Vorteile von Größe von Bytecode und Geschwindigkeit des Binärcodes zu kombinieren. Auf diesbezügliche Einzelheiten wird im Kapitel 4.2.1 eingegangen.

Es ist klar, daß jede Kombination dieser Strategien für die Implementierung einer Internetprogrammiersprache potentiell geeignet ist, und es gibt tatsächlich eine breite Streuung unter den aktuell implementierten Systemen. Betrachtet man die Vielzahl an unterschiedlichen Betriebssystemen und Hardwareplattformen, birgt die Möglichkeit der plattformunabhängigen Programmierung große Vorteile, da nur eine Realisierung einer Anwendung betrachtet werden muß. Die weite Verbreitung von HTML und jetzt auch von JAVA unterstützen diese These.

Obwohl Plattformunabhängigkeit schon lange ein Ziel der Entwickler von Programmiersprachen war, war stets die Nachfrage nach höchstmöglicher Performanz ein Entwicklungshemmnis (vor allem auf der Ebene des ausführbaren Codes). Im Internet tritt dieses Ziel aufgrund von ohnehin beschränkter Bandbreite in den Hintergrund, da hier die Transferzeit die primäre Komponente der Performanz einer Applikation ist und die Ausführungsgeschwindigkeit in den Hintergrund drängt. Dieser Faktor wurde in dem noch folgenden Performanzvergleich berücksichtigt (vgl. Kapitel 4.4.1). Der Vorteil, den das Versenden von Bytecode (siehe JAVA) gegenüber Quellcode (siehe

HTML) mit sich bringt, ist, daß eine Vielzahl von Programmiersprachen mit unterschiedlichem Quellcode eine entsprechend hohe Zahl von Compilern/Interpretern für den Quellcode benötigen würde. Einheitliche Codeformate dagegen würden die Zahl der benötigten Virtuellen Maschinen reduzieren. Die Fähigkeit sichere und portable Applikationen in einem (vor-) kompilierten Format zu laden, bringt gleichzeitig den Schutz des intellektuellen Eigentums des Autors mit sich. Das einfache „Abschreiben“ von kleinen Quellcodesegmenten kommt mehr einem „Mogeln“ nahe, wobei der Aufwand der Dekompilation[3] schon den Eindruck von tatsächlichem Diebstahl vermittelt.

Das Laden von Binärcode (siehe ActiveX) aus ungesicherten Quellen und über unsichere Netzwerke ist gefährlich. Der Anwendung sollte nicht erlaubt werden, direkt auf das Klientensystem zuzugreifen und dort, unter Umständen, Schaden anzurichten. Auch wenn die Sprache auf eine „sichere“ Untermenge eingeschränkt wird, gibt es keine Garantie, daß ausschließlich diese Untermenge auch tatsächlich vor der Kompilation verwendet wurde.

Nachdem die Möglichkeiten der Distribution von Programmen aufgezeigt wurden, zeigt das Beispiel HTML, daß das Laden von Quellcode einer sicheren Sprache, und dessen Ausführung mit einem vertrauenswürdigen Interpreter, sicher ist. Es ist nicht möglich einen Klienten mit einem Virus zu infizieren, indem ein HTML Dokument geladen und angezeigt wird (das Laden von virulenten Binärdateien birgt diese Gefahr natürlich, wird von HTML aber nicht unterstützt). HTML ist nicht mächtig genug - die grundlegende Idee ist aber von vielen Internet-programmiersprachen übernommen worden, die im Vergleich mächtiger aber nicht so weit verbreitet sind.

HTML ist für die Entwicklung von potenten, verteilten Anwendungen im Internet nicht geeignet, und auch nicht für diesen Zweck entwickelt worden. Schon gar nicht kann auf dieser Basis ein Modell verteilter Objekte aufgebaut werden. Die Möglichkeit der einfachen Entwicklung von Anwendungen auf Basis des Internets, mit optimaler Segmentierung zwischen Klienten und Server, im Kontext des heterogenen und dynamischen Umfeldes des Internets, ist Ziel der aktuellen Entwicklungsversuche [OSC97]. Dabei dürfen die Aspekte der Sicherheit, Performanz und der Schutz von intellektuellem Eigentum aber nicht außer acht gelassen werden.

2.3.3 Sockets

1982 wurden sogenannte *Sockets* (~ Fassung) als Schnittstelle für die (lokale) Interprozeß-kommunikation in dem 4.1c-BSD-System von „Berkley Software Distribution“ als generische Programmierschnittstelle eingeführt. 1985 führte SUN Microsystems sein „Network File System (NFS)“ und RPC auf Basis von Sockets ein. 1986 erweiterte die UNIX-Version 4.3 BSD die Sockets Schnittstelle für den Einsatz über TCP/IP und XNS Protokolle. Seitdem definiert man einen *Socket* als Kombination aus IP-Adresse und einem Port. Dieses Konstrukt ermöglicht die Kommunikation, auch von Prozessen auf verteilten Systemen, über das Internet. Breite Akzeptanz machten diese Schnittstelle zu einem Standard bei Netzwerken in der UNIX Umgebung.

Da es bis heute keine standardisierte Schnittstelle für Sockets unter MS Windows Betriebs-systemen gibt, entwickelte ein Konsortium von Firmen unter der Leitung von MS das WinSock „Application Programming Interface (API)“. Diese Programmierschnittstelle definiert die Bibliotheksaufrufe und die damit verbundene Semantik für die Programmierung von Sockets auf MS Windows Plattformen. Dabei wird eine Schnittstelle zwischen Programmen und dem verwendeten Transportprotokoll angeboten, die als bidirektionaler Nachrichtenkanal (*Pipe*) für ein- und ausgehende Daten fungiert. Die WinSock Implementierungen können wiederum verschiedene Netzwerkprotokolle wie TCP/IP, IPX/SPW, AppleTalk, NetBIOS, DecNet und

[3] Dies ist z.B. durch Werkzeuge wie MOCHA von Hanpeter van Vliet bei JAVA Bytecode möglich (http://www.oriondevel.com/robraud/classinfo/mocha.html).

andere unterstützen [SIN97]. Die WinSock API 1.1 ist konsistent zur BSD 4.3 Socket Schnittstelle und bietet weiter MS Windows spezifische Methoden zur Anwendungsprogrammierung auf Basis von Sockets an.

Es gibt also zwei Definitionen für den Begriff „Socket", die wie folgt differenziert werden sollen:

*Ein **Socket** ist eine Kombination aus IP-Adresse und Port mit entsprechendem API.*

***WinSock** ist ein Industriestandard für die Socket Programmierung unter MS Windows.*

Im weiteren soll unter dem Begriff *Socket* sowohl die Definition selbst, als auch eine WinSock Verbindung auf Basis von Sockets verstanden werden. Eine Trennung ist so nicht notwendig.

Aus der Sicht eines Programmierers vereinfachen Sockets die Netzwerkkommunikation stark. Dabei gibt es drei Arten von Sockets:

1. Datagram Sockets auf Basis des verbindungslosen UDP.
2. Stream Sockets auf Basis des verbindungsorientierten TCP.
3. Raw Sockets auf einer sehr niedrigen Schicht eignen sich vor allem zum Testen.

Verbindungsorientierte Protokolle erlauben eine zuverlässige, bidirektionale Kommunikationsverbindung innerhalb einer Sitzung. Jedes ausgetauschte Informationspaket erhält eine eindeutige Sequenznummer und wird individuell quittiert. Diese „Sicherheit" in der Kommunikation erfordert allerdings einen erheblichen *Overhead* für den Aufbau und für die Verwaltung der Sitzung. Verbindungslose Protokolle, wie Datagram Sockets, bieten eine einfache, aber unsichere Form der Kommunikation. Mächtigere Protokolle wie NetBIOS stellen erweiterte Broadcast Fähigkeiten (Multicast und Broadcast) zur Informationsverbreitung zur Verfügung. Datagramme werden vor allem für schnelle Informationsverbreitung verwandt [ORF97]. Diese unterschiedlichen Voraussetzungen führen dazu, daß Stream Sockets ca. 150 mal langsamer als Datagram Sockets sind [ORF97].

Die Verwendung von Sockets für die Implementierung verteilter Objekte ist, da das Socket Konzept auf einer sehr tiefen Schicht operiert, sehr aufwendig. Insbesondere die Verwaltung von vielen Klienten und Verbindungen führt zu erheblichem Programmieraufwand. Da für eine sichere Kommunikation auf Stream Sockets zurückgegriffen werden muß, ist dieses Modell auch sehr langsam. Die folgende Tabelle faßt die Eigenschaften von Sockets zusammen und dient am Ende dieses Abschnittes für den Vergleich der vorgestellten Technologien.

Eigenschaft	***Datagram***	***Stream***
Sichere Verbindung	nein	ja
Parameter Marshalling	nein	nein
Schnittstellenbeschreibung	nein	nein
dynamic discovery / dynamic invocation	nein	nein
Garbage Collection	nein	nein
Zugangskontrolle	nein	nein
Sicherheit	ja (SSL)	ja (SSL)
Einsatz durch Firewalls	ja	ja
proprietär	nein	nein

Tab. 1: Eigenschaften von Sockets [ORF97].

Diese Tabellenstruktur wird in den folgenden Kapiteln wiederholt, um dem Leser eine vergleichende und übersichtliche Darstellung der präsentierten Technologien zu ermöglichen. Bei Bedarf werden einzelne Punkte zugefügt oder ausgelassen.

Im besonderen bedeuten die aufgeführten Eigenschaften:

- Parameter Marshalling: Plattformunabhängige Übertragung von Methodenparametern.
- Schnittstellenbeschreibung Öffentlich verfügbare Beschreibung der Schnittstellendefinition.

- Dynamic Discovery — Das dynamische Auffinden von Methodenservern.
- Dynamic Invocation — Das dynamische Aufrufen von Methoden zur Laufzeit
- Garbage Collection — Möglichkeit der automatischen und verteilten Speicherbereinigung.
- Zugangskontrolle — Authentifizierung und Autorisation des Anwenders.
- Sicherheit — Schutz der übertragenen Information vor Dritten.

2.3.4 HTTP und CGI/ISAPI

In der Hierarchie des OSI-Modells [TAN97] operiert das „Hypertext Transfer Protokoll (HTTP)“ über Sockets. Vor der Einführung neuer Middleware-Technologien, war die Kombination des HTTP und des „Common Gateway Interface (CGI)“ das vorherrschende Modell für die Konstruktion von 3-Tier-Anwendungen[4] im Internet. HTTP [T. Berners-Lee, R. T. Fielding, H. Frystyk Nielsen] wird seit 1990 im Internet verwendet. Es spezifiziert wie statische Informationen zwischen einem Server und einem Klienten übergeben werden. Dadurch wird also ein Format von Klientenanforderungen und Serverantworten definiert und eine RPC ähnliche Semantik auf Basis von Sockets und TCP/IP zur Verfügung gestellt.

Die erste veröffentlichte Version 0.9 war ein einfaches Protokoll für einen einfachen Datenzugriff mit einem sehr beschränkten Befehlssatz.

Befehl	***Beschreibung***
GET	Anfordern der durch eine URL spezifizierten Daten
HEAD	Zurückgeben des HTTP Server Antwortkopfes
POST	Informationen zum HTTP Server senden um weitere Aktionen zu veranlassen
PUT	Senden von Daten, um diese auf dem Server zu speichern
DELETE	Löschen der durch eine URL spezifizierten Daten
LINK	Aufbau einer oder mehrerer Verbindungen zwischen spezifizierten URLs
UNLINK	Löst eine Verbindung zwischen zwei Ressourcen

Tab. 2: HTTP-Methoden für Klienten [COO96].

HTTP/1.0 erweiterte das Protokoll durch Einführung von selbstbeschreibenden Nachrichten unter Verwendung einer Variation des „Multipurpose Mail Extension (MIME)“ Protokolls[5] [BRU97]. Dadurch kann der Klient den Server über die von dem Klienten verstandenen Datenformate informieren. Im November 1995 wurde die vorläufige Version HTTP/1.1 veröffentlicht. Diese Version ist abwärtskompatibel zu HTTP/1.0 führt aber stärkere Reglementierungen ein, um sicherzustellen, daß die Eigenschaften des Protokolls zuverlässig implementiert werden.

Das HTTP Protokoll basiert auf einem Request/Response Paradigma. Der anfordernde Klient etabliert eine Verbindung mit einem Server und sendet einen Request (bestehend aus Request Methode, URL, Protokollversion und MIME-ähnlicher Nachricht) zum Server. Der Server antwortet mit einer Statusmeldung (Protokollversion und Erfolgs-/Fehlermeldung) gefolgt von einer MIME-ähnlichen Nachricht (Serverinformationen und möglichem Inhalt). Unter TCP/IP ist der vorgegebene TCP-Port 80, andere können aber konfiguriert werden, um zum Beispiel von einer Firewall verarbeitet zu werden. HTTP ist zustandslos; für jede neue Anforderung des Klienten wird eine neue Verbindung zum Server aufgebaut. Diese Eigenschaften resultieren zwar in eine einfache Implementierung, machen das Protokoll selbst aber nicht effizient. Dazu trägt weiter der Overhead der selbstbeschreibenden Nachrichten bei. Abschließend ist das Protokoll

[4] Eine dreischichtige Softwarearchitektur, die Datenbanksysteme in „externe Sicht“, „logisches Modell“ und „internes Modell“ aufteilt [RIC90].

[5] Der Internet Standard MIME für Nachrichteninhalte ist in den [RFC 1521] und [RFC 1522] spezifiziert.

durch seine Zustandslosigkeit für Einsätze in großen Klient/Server Architekturen und als Plattform für verteilte Objekte ungeeignet.

CGI ist die Schnittstelle zwischen einem HTTP-Server und einer Anwendung. Dadurch können Methoden (z.B. Datenverarbeitung und -suche) jenseits der Möglichkeiten von HTML und HTTP auf Serverseite ausgeführt werden. So kann zum Beispiel über CGI eine Anmeldung an eine Datenbank erfolgen, ein Datensatz gesucht und als HTML-Seite aufbereitet, über HTTP, an den anfordernden Klienten zurückgegeben werden. CGI ist auf unterschiedlichen Plattformen implementiert und wird oft als Skript bezeichnet, da es zumeist in Skript- oder Batchsprachen wie Perl, TCI und MS-DOS Batch realisiert ist [ORF97].

Neben der Erweiterung der Mächtigkeit von HTTP durch CGI, kommt für den Einsatz im offenen Internet der Sicherheit des Gesamtmodells eine besondere Bedeutung zu. An der Schnittstelle zwischen Klienten und Server bietet das „Secure Socket Layer (SSL)“[6] eine Möglichkeit der sicheren Kommunikation (vgl. Anhang B). SSL ist ein low-level Protokoll, das sichere Kommunikation zwischen WWW-Servern und Browser gestattet [DRAFT95]. In diesem Zusammenhang fällt auch der Begriff des „Hypertext Transfer Protocol with privacy (HTTPS)“. Eine HTTPS Verbindung ist eine HTTP Verbindung unter Verwendung der SSL-Verschlüsselung. Eine Beschreibung des Verbindungsaufbaus wird im Anhang B exemplarisch dargestellt.

Damit sind die Grundlagen für den „aktuellen“ elektronischen Markt geschaffen. Klar wird aber, daß die vorgestellte Architektur die Einfachheit weit über die Effizienz legt. Die Kombination von HTTP und CGI ist sehr langsam und durch die Zustandslosigkeit als „Middleware“ ungeeignet. Weiter bietet es, ohne Kombination mit HTTPS, keine inhärente Sicherheit und unterstützt damit auch keine Transaktionen. Trotz dessen ist diese Kombination das zur Zeit meist verwendete Modell für 3-Tier-Anwendungen im Internet [ORF97]. Aufgrund der weiten Verbreitung ist diese detaillierte Betrachtung von CGI sinnvoll. CGI ist eine Möglichkeit, die es einem Browser ermöglicht, eine Anwendung auf dem WWW-Server auszuführen. Entweder werden CGI-Anwendungen in einer Skriptsprache formuliert und zum Beispiel durch einen Perl Interpreter ausgeführt, oder in C/C++ geschrieben und in eine Anwendung kompiliert. Letzteres hat unter WinNT erhebliche Vorteile in der Performanz, da die binäre DLL nur einmal in den Adreßraum geladen werden muß.

Im Unterschied zu diesem Ansatz verwendet der NT Server von Microsoft nicht CGI, sondern das „Internet Server Application Programming Interface (ISAPI)“. ISAPI-Anwendungen sind DLLs anstelle von ausführbaren Programmen. Die ISAPI-DLL ist ein Kommunikationskanal zwischen dem „Internet Information Server“ und dem Internetdienst. Dieser WWW-Dienst lädt die ISAPI-DLL bei Bedarf in den eigenen Adreßraum. Eine ISAPI-Anwendung kann weiter die Vorteile der Win32-API und darüber auch die des „Open Database Connectivity (ODBC)“ Standards ausnutzen. Darüber hinaus bieten diese Anwendungen bessere Performanz, da sie die Vorteile von Speicherzeiger ausnutzen können, und nicht für die Beantwortung wiederholter Anforderungen neu gestartet werden müssen. Die DLL wird erst wieder aus dem Arbeitsspeicher entfernt, wenn sie lange nicht angefordert wurde. Dadurch entfällt auch der Prozeßwechsel auf der WWW-Serverseite, da die DLL in den Adreßraum des anfordernden Prozesses, also des WWW-Servers, geladen wird. Wenn die ISAPI-DLL geladen ist, ist der Code Teil des WWW-Dienstes und die ISAPI-Anwendung dient lediglich als Erweiterung [MS963]. Der „Internet Information Server“ der Firma Microsoft arbeitet mit zwei unterschiedlichen Typen von ISAPI-DLLs:

1. ISAPI-Anwendungen sind Erweiterungen, welche die dynamische Inhaltserstellung und bei Anforderung auch die explizite Inhaltserstellung ermöglichen. Explizite Seiten sind dabei auf dem WWW-Server abgelegt, dynamische Seiten werden erst bei Anfrage durch den Klienten erstellt.

[6] Vergleiche auch http://home.netscape.com/newsref/std/SSL.html.

2. ISAPI-Filter fügen sich in den Kommunikationspfad zwischen Klient und Server ein. Sie modifizieren oder observieren alle Ein- und Ausgaben und können das Verhalten des Servers daraufhin anpassen.

Der Einsatz von ISAPI wird in der folgenden Abbildung deutlich:

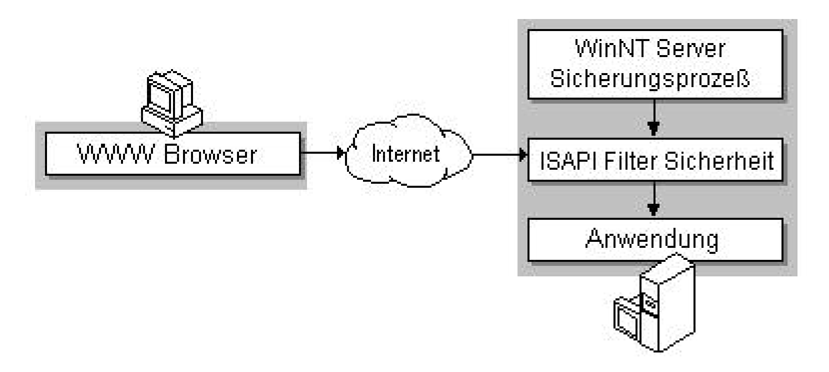


Abb. 5: ISAPI-Filter als weitere Sicherungsschicht.

Wie die Abbildung zeigt, können insbesondere die ISAPI-Filter auch einen Sicherheitsbeitrag bieten. Trotz der Vorteile, welche die ISAPI Technologie gegenüber dem weitaus stärker verbreiteten CGI bietet, ist die ISAPI Technik proprietär und bietet auch nur eine statische Darstellung von Informationen. Für Einzelheiten über Technologie und Anwendung von ISAPI siehe auch [MS963].

Eigenschaft	***HTTP/CGI***	***HTTP/ISAPI***
Sichere Verbindung	ja	ja
Parameter Marshalling	ja	ja
Schnittstellenbeschreibung	nein	nein
dynamic discovery / dynamic invocation	nein	nein
Garbage Collection	nein	nein
Zugangskontrolle	nein	ja
Sicherheit	ja (SSL, SHTTP)	ja (SSL, SHTTP, NT Sicherheit)
Einsatz durch Firewalls	ja	ja
proprietär	nein	ja

Tab. 3: Eigenschaften von CGI/HTTP [ORF97] und HTTP/ISAPI.

Messungen der Performanz in [ORF97] haben ergeben, daß die Kombination CGI/HTTP ca. doppelt so langsam wie Sockets sind. Im TCP/IP-Modell [HAL97] baut TCP in der Transportschicht auf IP in der sogenannten Internetschicht auf. Diese Kombination wird wiederum vom HTTP in der Anwendungsschicht verwendet - eine schlechtere Performanz ist dabei zwangsläufig. Der Einsatz von ISAPI kann diese Meßwerte besonders bei wiederholten Zugriffen verbessern, das Gesamtverhalten des Modells bleibt aber unzureichend.

2.3.5 Remote Method Invocation (RMI)

Am 3. Dezember 1996 kündigte JavaSoft seine „JAVA Development Kit (JDK) 1.1“ an. Teil dieser Entwicklungsumgebung ist die „Remote Method Invocation (RMI)“, welche ein verteiltes Objektmodell in die JAVA Sprache einführt. Obwohl die objektorientierte Programmiersprache JAVA noch in den folgenden Kapiteln zu diskutieren ist, soll hier RMI separat betrachtet werden.

RMI ermöglicht Programmierern von JAVA die Erstellung von verteilten JAVA Applikationen, in denen Methoden entfernter JAVA Objekte von JAVA „Virtual Machines“ aufgerufen werden können. Die „Virtual Machine (VM)“ interpretiert JAVA Bytecode und führt diesen dann auf dem Klientensystem aus. Die VM kann also in diesem Kapitel als Laufzeitumgebung von JAVA betrachtet werden. Eine nähere Diskussion erfolgt im Kapitel 4.2.

Ein JAVA-Programm kann ein entferntes Objekt aufrufen, sobald es eine Referenz auf das Objekt, entweder durch Verwendung des RMI „bootstrap“ Naming Service angefordert, oder als Argument oder Rückgabewert erhalten hat. Dabei kann ein Server, der ein solches Objekt veröffentlicht, auch selbst Klient sein. RMI verwendet die Methode für „Objekt Serialisierung“ des JDK V1.1, um ein lokales JAVA Objekt in einen seriellen Datenstrom („Marshalling Stream“) zu wandeln (*marshal*), innerhalb einer Nachricht über das *Remote Reference Layer* (vgl. Abb. 6) an einen Klienten zu übertragen und dort wieder zurück zu wandeln (*unmarshal*). Der empfangende Server-Skeleton entpackt den Marshalling Stream und übergibt die Argument an die aktuelle Implementierung der aufgerufenen Operation [MER97]. Für das eigentliche Verbindungsmanagement ist das *Transport Layer* verantwortlich, welches auf JAVA-Sockets und TCP realisiert ist. Im Unterschied zu JAVA (und C++) übergibt RMI lokale Objekte als Wert („by-value“), weil eine Referenz auf ein lokales Objekt nur innerhalb einer einzigen VM sinnvoll ist. Typen werden nicht gekürzt, so daß RMI echten objektorientierten Polymorphismus unterstützt [ORF97].

Der Namensdienst von RMI ist zur Zeit noch sehr primitiv, nicht persistent und dient lediglich zum Auffinden von aktiven Objekten (rmi://*hostname:port*/*name*). Er stellt weiter einen Mechanismus zur Verfügung, mit dem ein Klient Stubs vom Server laden kann. Wie auch bei anderen verteilten Objektmodellen dient im RMI ein *Stub* [RED96] als lokales Stellvertreterobjekt (*Proxy*) für ein entferntes Objekt (vgl. Abb. 6). Dabei bietet RMI einen, gegenüber JAVA, erweiterten Sicherheitsmechanismus, der das Wohlverhalten der Stubs überwacht. Abschließend erweitert RMI die Fehlerbehandlung von JAVA mit der `java.rmi.RemoteException` Klasse, um auch entfernte Fehler behandeln zu können.

Der Lebenszyklus entfernter Objekte wird durch einen Referenzzähler (sog „Live Reference“ [MER97]) realisiert, der die aktiven Verbindungen zwischen Objekten zählt. *Aktive Verbindungen* sind im Fall von JAVA Klient/Server-Verbindungen auf TCP/IP Basis. Sobald der Referenzzähler eines Objektes auf Null gefallen ist, wird dieses durch die verteilte „Garbage Collection“ von RMI gelöscht. Dieser Mechanismus ist zwar einfach zu implementieren und entwickelt zur Laufzeit kaum Overhead, Unterbrechungen aktiver Verbindungen aufgrund von Ausfällen auf unteren Protokollebenen führen aber leicht zu Inkonsistenzen. Automatisches *Rebinding*, das erneute Anbinden an entfernte Objekte nachdem eine Verbindung unterbrochen wurde, wird von RMI nicht angeboten [ORF97].

Die folgende Abbildung verdeutlicht das RMI-Modell.

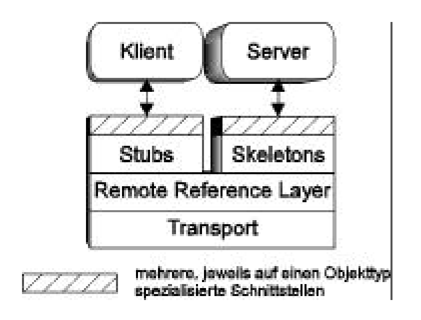


Abb. 6: RMI-Modell [MER97].

RMI erlaubt es, entfernte Objekte mit Hilfe von normalen JAVA-Schnittstellen zu erstellen. „Wie in CORBA oder DCOM sind also Vermittlungsschichten an der Kommunikation beteiligt, die der „RMI-Compiler“ *rmic* aus der Schnittstellenbeschreibung automatisch generiert [MER97].“ Dieses Umgehen einer eigenen Schnittstellenbeschreibungssprache führt automatisch dazu, daß RMI nicht Objekte adressieren kann, die in anderen Sprachen erstellt worden sind. Da RMI auf der Übertragungsschicht keine Unterstützung für Sicherheit und Transaktionen bietet, muß man dieses Kommunikationsprotokoll als proprietär und „lightweight“ bezeichnen.

Zur Zeit stehen zwei veröffentlichte Versionen zur Verfügung [SUN296]:

1. JDK 1.1 (http://java.sun.com/products/JDK/1.1/):
   Kernunterstützung für RMI. RMI Anwendungen (genaugenommen Applets, siehe Kapitel 4.2) können lediglich im JDK ausgeführt werden, solange Lizenznehmer nicht die VM des JDK 1.1 in ihren Browser Produkten implementiert haben. Dies ist zur Zeit noch nicht erfolgt.
2. RMI auf Basis des JDK 1.0.2 (http://chatsubo.javasoft.com/current/rmi/index.html):
   Diese Zwischenversion löst Betaversionen von RMI ab. Sie ist sowohl in der VM des JDK 1.0.2., als auch in besonderen Versionen von HotJava und Netscape 3.01 lauffähig.

Bei der Anwendung von RMI auf Servern, die jenseits einer Circuit-Level Firewall (siehe Kapitel 2.4.3.2) liegen, durch die also keine HTTP Anforderungen direkt in das Intranet weitergegeben werden (in diesem Fall wäre einfaches HTTP-Tunneling möglich [MER97]), muß ein CGI-Programm auf dem HTTP Server installiert werden. Dieses Programm stellt Port 80 stellvertretend für den „normalen" WWW-Server zur Verfügung und leitet HTTP-Anforderungen an den RMI-Server auf der Intranetseite weiter. Das CGI-Programm fungiert also als Stellvertreter für den WWW-Server. Das entsprechende Skript ist auf den Internetseiten von JavaSoft zu erhalten. Alternativ kann die Firewall auch als Application-Level Gateway auf dem RMI Port 1099 betrieben werden.

Eigenschaft	***RMI***
Parameter Marshalling	ja
Parameterübergabe [in, out, in/out]	in
Laden von dynamischen Stubs / Klassen	ja
Garbage Collection	ja
Schnittstellenbeschreibung	ja (JAVA)
dynamic discovery / dynamic invocation	nein
Sicherheit auf Transportebene	nein
Sicherheit auf Transaktionsebene	nein
Persistente Namensgebung	nein
Persistente Objektreferenzen	nein
Sprachneutrales Transportprotokoll	nein
Zugangskontrolle	nein
Einsatz durch Firewalls	ja
proprietär	ja
Überwachung der Lebenszeit	Referenzzähler

Tab. 4: Eigenschaften von RMI [ORF97].

Nachdem sich RMI (gegenüber Sockets und HTTP/CGI) als erste „echte" Middleware erweist, kann die vergleichende Übersicht um typische Merkmale erweitert werden. Für den Vergleich zu anderen Plattformen sind dabei folgende Eigenschaften von besonderer Bedeutung:

- Parameterübergabe — Richtung, in der Parameter über eine Schnittstelle übergeben werden können.
- Überwachung der Lebenszeit — Beantwortet die Frage wann eine Serverkomponente deaktiviert werden kann, ohne daß ein aktiver Klient unterbrochen wird.

Am 1.8.1997 hat SUN Microsystems Abteilung JavaSoft angekündigt, das RMI Protokoll in die CORBA Spezifikationen der OMG zu integrieren. Die Firma Microsoft unterstützt die Ausführung von RMI-Anwendungen weder im JAVA „Software Development Kit (SDK)" 2.0, noch im InternetExplorer 4.0. Für den Netscape Navigator ab V3.01 liegt dagegen ein Plug-In vor. Der Netscape Communicator verfügt über einen entsprechenden Patch [MER97].

2.3.6 Möglichkeiten und Grenzen aktueller Technologien

In der Diskussion um verteilte Objekte müssen die vorhandenen Technologien für eine Standortbestimmung angesprochen werden, auch wenn diese nicht direkt zu einer klassischen verteilten Middleware gezählt werden, wie sie im folgenden Kapitel 3 vorgestellt wird.

Die Entwicklung von Klient/Server Systemen auf Basis von Sockets ist ein sehr aufwendiger Vorgang. Eine Unterstützung für objektorientierte Programmierung wird nicht geboten und der Entwickler wird voll in die Protokollimplementierung auf den niederen Schichten des TCP/IP einbezogen. Dies hat zur Folge, daß komplexe, auf Sockets implementierte VS automatisch proprietär sind [ORF97].

Die Kombination aus HTTP/CGI oder HTTP/ISAPI, die prinzipiell auf Sockets basiert, ist umständlich, zustandslos und sehr langsam. Vor allem die Zustandslosigkeit läßt dieses Modell für große verteilte Anwendung ungeeignet erscheinen. Nichtsdestotrotz ist dieses Modell die am häufigsten verwendete Plattform für 3-Tier-Systeme im Internet [ORF97]. Zusammenfassend kann über diese beiden Modelle gesagt werden, daß sie nicht zuletzt auch aufgrund ihrer mangelhaften Performanz nur für einen „*Passiv Dataview*“ in einer klassischen Terminal/Server Konfiguration geeignet sind. Daten können lediglich von einem Anbieter angefordert, empfangen und dann offline dargestellt werden. Auch wird die notwendige Sicherheit auf Transportebene nur über „zusätzliche“ Protokolle wie SSL und SHTTP erreicht, die sich zum heutigen Zeitpunkt noch nicht als Standard gefestigt haben.

RMI kann dagegen schon als Middleware für verteilte Objekte aufgefaßt werden. Es unterstützt eine Schnittstellenbeschreibungssprache durch seine Implementierung in JAVA, hat eine verteilte Garbage Collection und eine gute Performanz. RMI kann als „Objekt-BUS“ auf Basis von JAVA bezeichnet werden, der folgende positiven Eigenschaften einführt:

- Code kann mit Daten übertragen werden.
- Durch die Implementierungssprache JAVA wird sowohl ein Sicherheitskonzept für Code, als auch eine Schnittstellenbeschreibungssprache eingeführt.
- Objekte können „By-Value“ übergeben werden, was unter CORBA und COM nicht möglich ist (vgl. entsprechende Kapitel 3.2.1 und 3.2.3).

Vermißt werden allerdings persistente Namensgebung und Objektreferenzen, Sicherheit auf Transportebene und die Transaktionsunterstützung. Vor allem ist RMI aber proprietär, hat somit auch kein sprachneutrales Übertragungsprotokoll und ist in der Implementierung auf JAVA fixiert. Weiter ist es neu, nicht etabliert und selbst nicht standardisiert.

Als Basis für eine Plattform für verteilte Objekte in einem Unternehmen, sind die vorgestellten Technologien damit unzulänglich.

2.4 Sicherheit

2.4.1 Sicherheit in verteilten Systemen

Vor dem Aufkommen verteilter Systeme waren Daten und Programme auf einem System konzentriert. Dieses konnte leicht gegen verschiedenste Gefahren geschützt werden, indem die unmittelbare Umgebung gesichert wurde. VS lassen sich dagegen nicht derartig schützen. Vergleicht man jedes Teilsystem innerhalb eines VS mit einer Insel, muß sowohl die Insel selbst, also auch die Kommunikation zwischen den Inseln gesichert werden. Weiter müssen sich die Inseln gegenseitig vertrauen, um miteinander kommunizieren zu können.

Bevor man sich aber Gedanken über die Sicherung eines VS macht, muß man sich die Frage nach dem Sinn und Aufwand der Sicherung stellen. Die Industrie ist dabei die Vorteile von VS, wie Kostenreduktion, Distribution von Angriffspunkten, Kommunikation etc. zu erkennen und damit diese Systeme auch einzuführen. Damit ist die Sicherheit von VS weniger eine technische, als eine

wirtschaftliche Frage, die sich vor allem auf die Relation von Kosten und Risiko konzentriert [BRU97].

> ***Sicherheit** eines DV Systems läßt sich als dessen Zustand definieren, in dem Informationen vor nicht autorisiertem Zugriff hinreichend dadurch geschützt werden, daß die relevanten Bedrohungen auf ein akzeptables Maß reduziert werden.*

Die Sicherung eines VS muß also ein Kompromiß zwischen dem (wirtschaftlichen) Aufwand für die Sicherungsmaßnahmen und der Gefahr durch das verbleibende Restrisiko sein. Absolute Sicherheit wird nie erreicht werden können [BRU97]. Im folgenden Kapitel wird ein Bedrohungsmodell vorgestellt, mit dessen Hilfe die Notwendigkeit für Sicherheit in VS dargestellt werden kann. Weiter dient es als Differenzierungsinstrument für die noch vorzustellenden Middlewareplattformen. Nur durch das Verständnis der Bedrohung, kann später die Qualität und Praktikabilität der verwendeten Sicherungsmechanismen der Plattformen für verteilte Objekte beurteilt werden.

2.4.2 Ein Bedrohungsmodell

2.4.2.1 Die Bedrohung

Bei der Darstellung der Bedrohung gilt es zwischen den klassischen Bedrohungen und den neuen Bedrohungen zu unterscheiden, die sich aus der Öffnung und Vernetzung von EDV-Systemen ergeben.

Bei der *klassischen Bedrohung* steht ein (argloser) Benutzer eines EDV-Systems einer Bedrohung gegenüber. Diese kann aus einem, zumeist technisch versierten Mitarbeiter, der sich Arbeitserfolge aneignen will, einem Saboteur, einem (Industrie-) Spion oder sich selbst bestehen. Für eine Darstellung von Angriffsmethoden vergleiche [BRU97]. Nach einer Umfrage betrachten 80% aller Befragten Fahrlässigkeit und Irrtum als primäre Gefahrenquelle [KES94].

Die *neuen Bedrohungen* ergeben sich durch das Aufkommen vernetzter und verteilter Systeme. Die heutigen Netze, in denen Rechner wie intelligente Inseln große Menge von Daten und Programmen speichern und diese über das offene Internet austauschen, sind besonders durch ihre Transportstrecken gefährdet. Im Kontext dieser Arbeit ist das Internet als Transportstrecke ein hochgradig gefährdeter Übertragungsweg. Computer können unbefugt betreten, deren Speicher ausgespäht, Inhalte gestohlen, verändert oder vernichtet werden. Der Nachrichtenverkehr wird beobachtet, Nachrichten werden abgefangen, verändert oder später wiederverwendet.

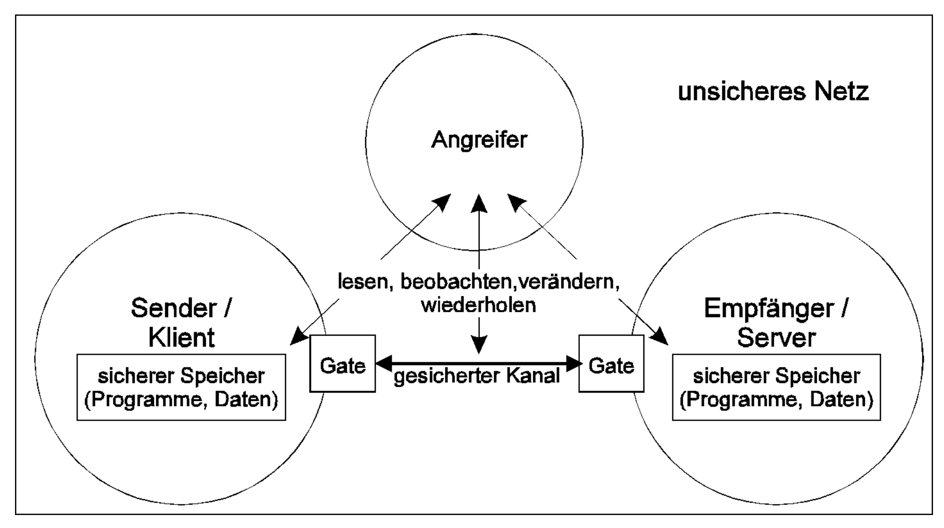


Abb. 7: Ein Bedrohungsmodell.

Nachdem die Systeme von Sender und Empfänger selbst geschützt wurden, muß auch der Kommunikationskanal zwischen diesen „Inseln“ gesichert werden. Die Verbindungspunkte der

Systeme zum unsicheren Netz, die sogenannten *Gates*, müssen die einzige Öffnung im Sicherungsnetz der Systeme sein.

Die verschiedenen Klassen von Bedrohungen werden in der folgenden Tabelle kurz zusammengefaßt. Für eine detaillierte Beschreibung siehe [BRU97].

Art der Bedrohung	***Beschreibung***
Diebstahl von Informationen	Aneignung von nicht öffentlichen Daten.
Zerstörung von Informationen	Zerstörung von wertvollen Daten. Diese können entweder selbst einen Wert haben (z.B. Kundenlisten) oder kostenintensiv in der Beschaffung sein (z.B. Umfrageergebnisse).
Diebstahl von Ressourcen	Benutzung von fremden Systemressourcen für eigene Zwecke, zum Beispiel Durchführung von komplexen Berechnungen.
Behinderung von Diensten	Dienste (z.B. Mail oder WWW-Server) durch übermäßigen Gebrauch für andere unbrauchbar machen.
Maskierung	Durch Vorgaukeln einer falschen Herkunft oder eines falschen Namens kann ein Programm einen Anwender auffordern, nichtöffentliche Daten einzugeben.
Tarnung	Zwischen Anwendungsprogramm und Datenbestand kann ein getarntes Programm den Datenfluß verfälschen, ohne daß dies von dem Anwender bemerkt wird. Dadurch können zum Beispiel Fehlentscheidungen durch den Anwender verursacht werden.

Tab. 5: Bedrohungsarten.

2.4.2.2 Angriffsarten

Von den dargestellten Bedrohungsarten können wiederum eine Vielzahl von verschiedenen Angriffsarten ausgeführt werden. Diese Angriffsarten werden in passive und aktive Angriffe unterteilt. *Passive Angriffe* dienen lediglich dem unerlaubten Informationsgewinn. Dazu gehören:

1. Abhören von Informationen
2. Abhören von Teilnehmeridentitäten
3. Analyse des Verkehrsflusses

Beim direkten Abhören von Informationen kann jede Art von Information wie zum Beispiel Paßwörter gewonnen werden. Das Abhören von Teilnehmeridentitäten läßt zumindest Rückschlüsse auf das Verhalten von Teilnehmern und unter Umständen, auch auf die Art der ausgetauschten Informationen zu. Die Analyse des Verkehrsflusses leistet dagegen keine semantische Untersuchung des Datenstroms, kann aber Informationen wie Häufigkeit, Zeitpunkte oder Größenordnungen von Kommunikationsverbindungen ermitteln, was zum Beispiel bei Börsentransaktionen einen großen Schaden verursachen kann. Eine besondere Gefährdung durch passive Angriffe besteht aufgrund des verwendeten Übertragungsprotokolls auch latent in Intranetumgebungen.

Passive Angriffe erfordern in der Regel einen physischen Zugang zum Übertragungskanal. Diese können durch bauliche Maßnahmen unterbunden werden. Dabei ist allerdings zu beachten, daß auch die Analyse kompromittierender Strahlung zum Informationsgewinn des Angreifers beitragen kann. Dazu gehören unter anderem elektromagnetische Strahlung der EDV Geräte (z.B. Monitore) oder die induktive Kapazität der Übertragungswege. Abhilfe können Maßnahmen zur Abschirmung und alternative Übertragungskanäle zum Beispiel aus Glasfaser bieten. Diese Abwehrmechanismen sind generell für alle DV Systeme zu treffen, auch wenn diese nicht verteilt sind. Das Vorhandensein vor allem der baulichen Maßnahmen und der Mechanismen zur Abschirmung wird in dieser Arbeit vorausgesetzt.

Im Unterschied zu passiven Angriffen versuchen *aktive Angriffe* Informationen zu vernichten oder zu modifizieren. Es handelt sich dabei um vorsätzliche Angriffe, die Informationen oder Kommunikationsverbindungen verfälschen, oder den Benutzer zu einem falschen Verhalten veranlassen.

Mögliche Angriffsarten sind:

1. Wiederholung oder Verzögerung einer Information,
2. Modifizierung von Information,
3. Vortäuschung einer falschen Identität,
4. Boykott eines Kommunikationsnetzes,
5. Nutzung fremder Betriebsmittel,
6. Leugnen der Teilnahme einer Kommunikation und
7. Schadprogramme.

Zu den bekanntesten Vertretern der aktiven Angriffe gehören die Viren, die erst durch die Verbreitung des Internets aufgekommen sind.

Für diese Form von Angriffen ist ein Zugang zum Netzwerk auf Protokollebene erforderlich, was zum Beispiel bei der Verbindung des Intranets eines Unternehmens an das öffentliche Internet der Fall ist. Einige dieser Strategien werden als Netzwerksniffing, Spoofing und Vulnerability Scan bezeichnet. Beim Netzwerksniffing werden Nachrichten, die in einem Broadcast-Netzwerk verschickt werden, abgehört und analysiert. Unter Spoofing versteht man die Vortäuschung einer falschen Identität, und der Vulnerability Scan bezeichnet das Suchen und Ausnutzen von system-immanenten Sicherheitslücken. Sicherheit bieten in diesen Szenerien die Maßnahmen zur Zugriffsbeschränkung und -vermeidung und Verschlüsselung, die im Gate realisiert werden. Hier wird auch versucht, das passive Abhören von Informationen zu verhindern.

Eine Beschreibung der verschiedenen Realisierungsmöglichkeiten erfolgt im Kapitel 2.4.3.

2.4.2.3 Sicherheitsstrategien

Die erste Gruppe von Gegenmaßnahmen besteht aus Zutritts-, Zugangs- und Zugriffsschutz des EDV-Systems. Diese Maßnahmen haben in der Umgebung von Großrechnern eine lange Tradition, werden aber nur langsam auf den Bereich der Personalcomputer übertragen. Die zweite, für diese Arbeit interessantere Gruppe, besteht aus Übertragungs- und Transportschutz.

Es gibt vier „Grundgüter“ der Sicherheit

1. Vertraulichkeit: kein Unbefugter soll Kenntnis über Daten erhalten
2. Integrität: kein Unbefugter soll Daten verändern können
3. Verfügbarkeit: kein Unbefugter soll Daten vernichten können
4. Verbindlichkeit: der Urheber von Daten soll immer erkennbar sein

Die Strategie besteht darin,

- Schäden möglichst zu vermeiden
- nicht vermeidbare Schäden möglichst schnell zu erkennen
- nicht vermeidbare Schäden möglichst zu begrenzen
- Beschädigungen beweiskräftig zu dokumentieren.

In der folgenden Matrix können die zur Verfügung stehenden Gegenmaßnahmen der entsprechenden Strategie zugeordnet werden.

<table>
<tr><td>Strategien</td><td colspan="2">Vermeidung</td><td colspan="2">Erkennung</td><td colspan="2">Begrenzung</td><td colspan="2">Dokumentation</td></tr>
<tr><td>Verfügbarkeits-verlust</td><td rowspan="3">Zugangs- und Zugriffs-schutz</td><td></td><td></td><td rowspan="3">Sicherheits-stand</td><td rowspan="4">Ala-mierung</td><td>Backup / Restore</td><td rowspan="4">Proto-kollierung</td><td></td></tr>
<tr><td>Vertraulickeits-verlust</td><td>Verschlüs-selung</td><td></td><td rowspan="3"></td><td></td></tr>
<tr><td>Integritätsverlust</td><td></td><td rowspan="2">elek-tronische Signatur</td><td></td></tr>
<tr><td rowspan="2">Verbindlichkeits-verlust</td><td colspan="2"></td><td rowspan="2"></td><td>Signierung</td></tr>
<tr><td colspan="3">Übertragungssicherheit</td><td colspan="3">Notariat</td><td>Archiv</td></tr>
</table>

Tab. 6: Sicherheitsstrategien und -maßnahmen.

Vor allem die Strategien der Erkennung, Begrenzung und Dokumentation müssen in einem DV-System vorausgesetzt werden können. Durch den Anschluß eines Unternehmenssystems an ein öffentliches Netz wie dem Internet, kommt der Vermeidung eine besondere Bedeutung mit einer neuen Qualität zu (siehe Kapitel 2.4.2.4). Abgesehen von den Schäden durch Fahrlässigkeit und Mutwillen der eigenen Mitarbeiter, kommen in diesem Szenario Gefahren durch mutwillig operierende „Hacker“ und konkurrierende Unternehmen hinzu.

Für eine begriffliche Präzisierung wird in der weiteren Diskussion ein Grundmodell verwendet, das sich wie folgt aufbaut: Ein Objekt, *Subjekt* genannt, stellt an ein *Objekt* eine *Anforderung* und das Objekt liefert an das Subjekt ein (auch negatives) Ergebnis. Das Objekt ist für seinen Schutz verantwortlich.

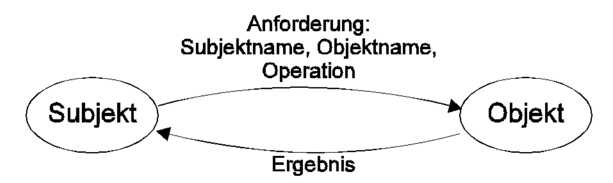


Abb. 8: Die Verantwortung eines Objektes.

Je nach Grad des Vertrauens zwischen Subjekt und Objekt können stärkere oder schwächere Schutzmechanismen angewandt werden, um den „Break-Even Point“ [COL89] bei der Kosten-Nutzen Funktion anzunähern [BER96].

2.4.2.4 Vermeidende Schutzmechanismen

Für Anwendungen im Internet haben die vermeidenden Schutzmechanismen also eine vorrangige Bedeutung. Die *vermeidenden Schutzmechanismen* haben die Aufgabe, ein bedrohliches Ereignis nicht entstehen zu lassen, d.h. der Angriffsversuch selbst kann zwar nicht unterbunden, sein Scheitern aber mit hoher Wahrscheinlichkeit herbeigeführt werden. Folgende Mechanismen finden besonders bei der Anwendung im Internet Einsatz:

- Zutrittsschutz — kein Unbefugter soll sich einem System annähern können
- Zugangsschutz — kein Unbefugter soll ein System benutzen können
- Zugriffsschutz — schützt Daten in einem System vor Manipulationen (durch ein Programm)
- Übertragungsschutz — Vermeidung der Übertragung manipulierter Daten

Die Subjekt-Objekt Beziehung wird in der folgenden Tabelle verdeutlicht:

Maßnahme	***Objekt***	***Subjekt***	***Anforderung***
Zutrittsschutz	Raum	Person	Betreten
Zugangsschutz	System	Person	Anmelden
Zugriffsschutz	Datei	Programm	Öffnen
Übertragungsschutz	Empfänger	Sender	Datenübertragung

Tab. 7: Vermeidende Schutzmechanismen.

Es ist klar, daß die Verantwortung für die Durchführung der Schutzmechanismen sinnvollerweise bei dem Objekt liegt. Aktionen des Subjektes lassen sich unabhängig von der Motivation nicht beeinflussen. Im Rahmen des verwendeten Grundmodells lassen sich leicht die folgende Aufgaben für das Objekt identifizieren:

1. Identifizierung: Ist der Subjektname ein korrekter Identifikator des Subjektes?
2. Authentifizierung: Beweis der Identität des Subjektes.
3. Autorisation: Darf das Subjekt die Tätigkeit „Operation" am Objekt ausführen? (Problem einer globalen Signatur)

Diese drei Teilschritte werden nur bei besonders kritischen Systemen bei jeder einzelnen Anforderung durchgeführt. Zwischenergebnisse dieser Teilschritte können auch gespeichert, oder Teilschritte zu einer Gruppe zusammengefaßt werden, um den Overhead des Verfahrens zu reduzieren. Die Realisierung dieser Schutzmechanismen erfolgt in der Regel durch einen Monitor, der jede Anforderung überwacht. Identität und Authentizität werden aufgrund von Datenbasen überprüft, die, je nach Organisationsmethode, als Zugriffsmatrix (Subjekt, Objekt und Tätigkeit als Dimensionen), „Capability List" (subjektorientiert) oder „Access Control List (ACL)" (objektorientiert) bezeichnet werden [BER96].

2.4.2.5 Vertrauen

Wie sich später im Vergleich der Middlewareplattformen noch zeigen wird, kommt bei der Frage nach der Sicherheit von verteilten Objekten dem Begriff des Vertrauens eine zentrale Bedeutung zu. Warum sollte man sich, seine Aufgaben oder Wirtschaftsprozesse einem System anvertrauen, welches man selbst nicht überschauen, eventuell nicht einmal örtlich bestimmen kann? Wenn nicht die einzelnen Komponenten des VS den persönlichen Ansprüchen genügen, wird man den Komponenten, und damit dem Gesamtsystem, nicht vertrauen.

> ***Vertrauen*** *ist das zuversichtliche Hoffen auf Integrität, Ehrlichkeit, Gerechtigkeit und Verfügbarkeit eines Objektes* [BRU97].

Vertrauen bezieht sich in der EDV auf die Fähigkeit von Anwendungen, Anforderungen integer durchzuführen, Informationen geheim zu behandeln und diese Aufgaben kontinuierlich durchzuführen. Im Gegensatz zu diesem allgemeinen Verständnis von Vertrauen, besteht das zweite Merkmal in der Verfügbarkeit des angeforderten Objektes. So muß ein Anwender grundsätzlich davon ausgehen können, daß zum Beispiel eine Transaktion tatsächlich in akzeptabler Zeit ausgeführt wird. Ist diese Prämisse nicht erfüllt, wird der Anwender die Benutzung des VS ablehnen. Verfügbarkeit impliziert bei einem VS dabei:

- Beständigkeit: Die Fähigkeit, vollständige (auch physikalische) Dienstunterbrechungen zu überstehen.
- Widerstandsfähigkeit: Die Fähigkeit, schleichenden oder teilweise auftretenden Dienstausfällen zu begegnen.
- Rettung: Die Fähigkeit, ohne Eingriff von außen, nach einer vollständigen Dienstunterbrechung den normalen Dienstbetrieb wieder aufnehmen zu können.
- Konsistenz: Die Fähigkeit, zu jeder Zeit die selben Ergebnisse auf eine Anfrage zu liefern. Der verteilte Prozeß muß jederzeit konsistent sein.

Das letzte Merkmal ist die Performanz eines vertrauenswürdigen VS. Damit ein VS benutzt und akzeptiert wird, muß es seine Aufgaben in einer der Anforderung entsprechenden Zeit, also so schnell wie nötig, erfüllen.

Vom Standpunkt der Sicherheit müssen Beschränkungen so eng wie möglich sein, aus der Sicht des Vertrauens müssen aber die Beschränkungen nicht die Leistungsfähigkeit und Verfügbarkeit des VS beeinträchtigen, ohne inakzeptabel hohe Risiken einzuführen.

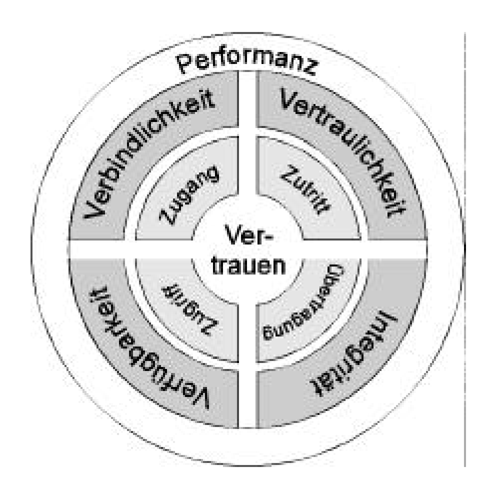


Abb. 9: Vertrauensmodell.

Aus der Diskussion um Vertrauen, in Verbindung mit den genannten Sicherungsstrategien, ergibt sich das Vertrauensmodell. Eine Hinzunahme von einem Sicherheitsmerkmal „Zugriffskontrolle“ (wie z.B. in [BRU97]) erscheint nicht sinnvoll, da die Zugriffskontrolle eine Methode zum Erreichen von Sicherheit und Vertrauen, weniger aber ein Merkmal oder eine Eigenschaft ist.

Dieses Vertrauensmodell umfaßt nicht alle aufgeführten Sicherheitsstrategien. Zum Beispiel ist der Zutrittsschutz eine unerläßliche Aufgabe in jedem Unternehmen, soll aber in der Diskussion hier nicht abgedeckt werden, weil es bei diesem Thema nicht um verteilte Objekte selbst, sondern um EDV-Sicherheit allgemein geht. Ähnliches gilt auch für Backup und Restore, Dokumentation, Archivierung etc. Das Augenmerk liegt also vorrangig auf den vermeidenden Schutzmechanismen.

Es ist klar, daß alle Komponenten des Vertrauens aufeinander aufbauen. Das Fehlen auch nur einer Komponente läßt die Vertrauenswürdigkeit eines Objektes sofort verschwinden und das Ziel eines VS obsolet werden. Die einzelnen Komponenten müssen aufeinander und mit der Firmenstrategie bezüglich Sicherheit abgestimmt werden. Nur so entsteht ein Gleichgewicht zwischen Freiheit und Restriktionen, das ein offenes, produktives und liberales Arbeitsklima schafft und gleichzeitig den notwendigen Schutz für das Unternehmen garantiert. Die Definition von *notwendigem Schutz* ist dabei stark von den individuellen Sicherheitsbedürfnissen des Unternehmens abhängig, muß aber unabhängig davon als Unternehmensstrategie formuliert sein. Dies ist unter anderem auch eine Forderung des „Orange Book“, das im folgenden Kapitel noch vorgestellt wird.

2.4.3 Methoden zum Schutz eines Intranet

2.4.3.1 Host Sicherheit

Ausgehend von der Erkenntnis, daß eine Sicherung beim Objekt zu beginnen hat, ist die Sicherung des Hosts selbst die letzte und gleichzeitig auch eine sehr schwache Bastion im Schutzmechanismus. Die Sicherheitsanforderungen werden im „Trusted Computer System Evaluation Criteria“ definiert. Dies ist eine Beschreibung der Anforderungen von Sicherheitsklassen, die durch das U.S. National Computer Security Center in einem Dokument, dem sogenannten „Orange Book“ [DOD85], festgehalten sind.

Die Klassen des „Orange Book“ beziehen sich mehr auf militärische und behördliche Anforderungen für Sicherheit von geheimen Daten, bieten aber einen Leitfaden auch für allgemeingültige Technologien. Viele kommerzielle Produkte, wie zum Beispiel das Betriebssystem WinNT, werden als „C2 Kompatibel“ bezeichnet. Dies bedeutet, daß sie dem geforderten Standard entsprechen, nicht aber die offizielle Überprüfung unterzogen wurden. Die einzelnen Klassen bauen von den Anforderungen jeweils aufeinander auf.

Klasse	***Beschreibung***
keine	Die Anforderungen der Klassen A, B oder C werden nicht erfüllt
C1	Discretionary Security Protection
C2	Controlled Access Protection
B1	Labeled Security Protection
B2	Structured Protection
B3	Security Domains
A1	Verified Design
A+(2)	Verified Implementation

Tab. 8: Klassen des „Orange Book“.

C2 wird hier als Minimalanforderung für einen sicheren Host eingestuft. Ein genaue Beschreibung der einzelnen Klassen findet sich in [BRU97] und [DOD85]. Trotz dieser hohen Anforderungen werden in der Realität auch auf die A1 kompatiblen Systeme keine hoch sensitiven Daten abgelegt, wenn unbekannte Anwender potentiellen Zugriff haben [CHES94]. Weiter ist, nach Definition, ein Netzwerkcomputer nicht isoliert. Damit vertrauen ihm zwangsläufig auch andere, verbundene Systeme. Hat ein Anwender eine Kennung mit ausreichenden Berechtigungen erlangt, kann er auf andere Systeme übergreifen.

Allein die Sicherung des Host ist also nicht ausreichend, da ein Einbruch hier ein untragbares Risiko mit erheblichen Folgerisiken birgt. Aufgrund des gegenseitigen Vertrauens der Computer muß ein Unternehmensnetz als Ganzes geschützt werden.

2.4.3.2 Firewall Gateways

Der Schutz eines Teilnetzes vor unbekannten, fremden Anwendern ist also ratsam. Eine Möglichkeit hierfür bietet eine Firewall, die per Definition die Abgrenzung eines Intranet leistet. An ihr werden Eindringlinge abgehalten, die eine potentielle Gefahr für das Intranet bieten:

> *„Eine **Firewall** ist eine Menge von Komponenten zwischen zwei Netzwerken, die insgesamt die folgenden Eigenschaften hat:*
> - *jeder Datenverkehr von einem Netz in das andere verläuft durch die Firewall*
> - *nur autorisierter Datenverkehr, durch Sicherungsregeln definiert, kann die Firewall passieren*
> - *eine Firewall selbst kann nicht penetriert werden* [CHES94]“

Diese „Menge von Komponenten“ setzt sich wie folgt zusammen:

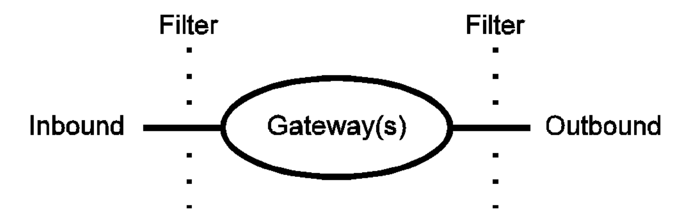


Abb. 10: Schematischer Aufbau einer Firewall [CHES94].

Die Komponenten einer Firewall sind in der Regel *Filter* (Screens), welche die Übertragung bestimmter Klassen von Datenverkehr unterbinden, und *Gateways*, welche die Auswirkungen der Filter kompensieren. Die praktische Realisierung von Firewalls erfolgt meist durch den Einsatz

von kommerziellen *Routern*[7], deren Routingtabellen als Filter programmiert werden. Diese Systeme werden zumeist auf dedizierten Computern betrieben.

Der Einsatz von Firewalls birgt folgende Vorteile:

1. Hosts im allgemeinen können sich, nicht zuletzt aufgrund ihrer Komplexität, nicht ausreichend schützen. Eine Firewall als dedizierter Host ist weniger anfällig für Angriffe, da sie kaum potentiell gefährliche Dienste (wie z.B. RLOGIN, TELNET und FTP) anbieten muß.
2. Firewalls werden in der Regel von dedizierten Administratoren gewartet. Dieser Personenkreis ist zumeist stark für potentielle Gefahren sensibilisiert.
3. Firewalls müssen, da sie dedizierte Systeme sind, nur ein äußerst geringen und bekannten Anwenderkreis bedienen (nur zur Wartung). Dadurch ist auch die Gefahr von „einfachen Paßwörtern" leichter zur reduzieren.

Durch den dedizierten Einsatz kann weiter darauf verzichtet werden, daß andere Maschinen einer Firewall vertrauen. Sie ist damit per Definition kein Netzwerkcomputer. Wird die Firewall von einem Eindringling übernommen, kann dieser nicht weitere, angeschlossene Systeme manipulieren, da diese der Firewall kein Vertrauen schenken. Dies ist die eigentliche Grundidee, die den Einsatz von Firewall so effizient und erfolgreich macht.

Häufig werden Firewalls auch als *Bastion Hosts* bezeichnet, die das *inbound* (innerhalb der Firewall) von dem *outbound* Netzwerk (jenseits der Firewall) trennen [CHES94]. Diese Namensgebung verdeutlicht die Funktion der Firewall als „sicheres Tor einer Festung".

Die verschiedenen Funktionen einer Firewall liegen in den Schichten 2 bis 7. Die folgende Abbildung veranschaulicht die Eingliederung von Firewalls in das OSI-Modell [BRU97].

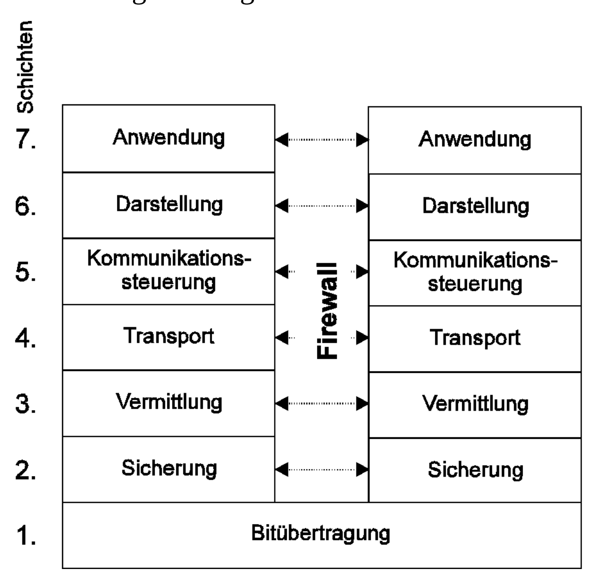


Abb. 11: Einbettung von Firewalls in das OSI-Modell.

Dadurch wird ersichtlich, daß eine Firewall ab Schicht 2 jeglichen Protokollverkehr zweier kommunizierender Schichten kontrolliert (siehe dazu und zum TCP/IP-Modell [TAN97]).

Es gibt drei Grundtypen von Firewalls:

1. *Paket Filtering Gateways*
   Diese Gateways bieten eine einfache und billige Möglichkeit der Sicherung. Die Fähigkeit der Filterung resultiert aus dem Einsatz eines Routers, der zur Verbindung eines Intranet zum Internet ohnehin benötigt wird. Paketfilternde Gateways selektieren Pakete aufgrund von

[7] Verbindungssysteme zwischen TCP/IP-Netzen, die auch Vermittlungsfunktion erfüllen [HER93].

Quell- oder Zieladresse und Port. Unerwünschte, also potentiell unsichere Pakete, werden nicht übertragen. Die Konfiguration der Routen ist allerdings ein für Fehler sehr anfälliger Vorgang, was den ausschließlichen Einsatz dieser Gateways nicht empfehlenswert macht [CHES94]. Zudem können Manipulationen nur schwer erkannt werden.

2. *Application-Level Gateways*
   Die Gateways auf Anwendungsschicht sind das Gegenstück zu den Paketfiltern. Anstelle einer Regelung auf sehr niedriger Ebene, die unabhängig vom Inhalt des Datenstroms operiert, werden hier nur sehr dedizierte Anwendungen freigegeben. Dies ist schon allein deswegen sinnvoll, da hier kaum Fehler durch falsche Konfiguration möglich sind. Es muß nur eine (kleine) Menge von Anwendungen administriert und auf ihre Sicherheit hin überwacht werden. Darüber hinaus können bei diesen Gateways alle ein- und ausgehenden Daten protokolliert werden. Diesen Vorteil bieten paketfilternde Systeme nicht, da hier schon vor einer möglichen Protokollierung Daten abgelehnt werden müssen. Eine Protokollierung auch dieser Daten würde eine neue Sicherheitslücke eröffnen und ist in der strikten Definition einer paketfilternden Firewall auch nicht möglich.
   Häufig werden zum Beispiel Mail Gateways als Application-Level Gateways, unabhängig von anderen eingesetzten Technologien, realisiert, da dieses Vorgehen keine Manipulation am vorhandenen Mail-System erfordert [CHES94]. Eine Konfiguration einer Middlewareplattform (wie z.B. CORBA und COM) als Klient für ein solches Gateway ist in einigen Fällen möglich, auf die im weiteren noch eingegangen wird.

3. *Circuit-Level Gateways*
   Diese Gateways basieren direkt auf TCP-Verbindungen. Ein Anrufender verbindet sich mit einem TCP-Port auf dem Gateway, dieses stellt wiederum eine Verbindung zu einem (anderen) TCP-Port auf der anderen Seite der Firewall her und „vermittelt“ den Datenstrom so durch die Firewall, die wie ein Verbindungskabel (circuit) agiert. In diesem Vermittlungsverfahren wird in der Regel der Datenstrom selbst nicht observiert [CHES94]. Gerade Middlewareplattformen auf Basis von TCP/IP (wie in der Beispielimplementierung vorgestellt) sind für Circuit-Level Gateways geeignet.

Diese Grundtypen können zu sehr wirkungsvollen Sicherungsmechanismen kombiniert werden und ein Intranet effektiv gegen nicht autorisierte Zugriffe aus dem Internet sichern. Die Sicherung erfolgt dabei auf einer sehr niedrigen, nicht inhaltlichen Ebene. So kann zum Beispiel ein unbekannter Virus, der in einer Mail enthalten ist, nicht von einer „klassischen“ Firewall erkannt werden.

Firewalls sind ein starkes Werkzeug zum Schutz von Datennetzen. Dennoch bieten sie nur Schutz mit beschränkten Möglichkeiten. Es ist also auch wichtig zu wissen, was Firewalls nicht leisten, respektive wo ihre Grenzen liegen. Ausgehend von den üblichen Schichten eines Netzwerks (Schichten 2-4 des OSI-Modells), stellt die Firewall einen guten Schutz dar. Spoofing oder verbotene Dienste können relativ problemlos erkannt werden. Falls eine Attacke jedoch auf höherer Ebene ansetzt, muß die Firewall deren Paketinhalt durchsuchen. Die Überlegung, kritische Dienste von vornherein zu sperren, ist also angebracht. Ein bekanntes Beispiel ist SendMail: Beim Interpretieren des Inhalts von bestimmten Mail-Headern, läßt sich dieser Mail Klient (in einer bestimmten Version) zu unerwünschten Aktionen mißbrauchen. Dies verdeutlicht, daß in Fällen, bei denen der Programmcode einer Anwendung schon fehlerhaft und unsicher ist, auch die beste Firewall nutzlos wird [CHES94]. Eine große, resultierende Einschränkung ist, daß sowohl JAVA als auch ActiveX Programme nicht von einer einfachen Firewall analysiert werden können. Das sogenannte „Exploder Objekt“ (siehe dazu auch Kapitel 4.3.2) kann nicht als unerwünscht identifiziert und ein JAVA-Applet, das eine Lücke in der VM gefunden hat [MOR97], nicht abgelehnt werden. Eine „klassische“ Firewall, die ein Intranet definiert, kann heute nur so konfiguriert werden, daß JAVA und/oder ActiveX-Komponenten durchgelassen werden oder nicht.

Eine sehr neue Entwicklung sind die sogenannten *intelligenten Firewalls*. Viele Entwickler von Firewalls glauben, daß eine Überwachung der komplexer werdenden Datenströme nur durch verbesserte Technologien der Analyse erfolgen kann. Ein Vorreiter mit einem Marktanteil von ca. 44% ist die Firma Checkpoint Software, die der Frage nach JAVA /ActiveX-Sicherheit mit ihrem Produkt „Firewall-1“ beantwortet [CHP97]. Dabei wird der Versuch unternommen, das Verhalten von verteilten Objekten bzw. Anwendungen auf Basis von Kontrollentscheidungen zu analysieren. Das sogenannte „Inspection Module“ liegt zwischen den Schichten 2 und 3 des OSI-Modells und kann so alle übertragenen Pakete analysieren, bevor diese das Betriebssystem erreichen. Entscheidungsgrundlage sind der Kommunikationsstatus, der von vorherigen Verbindungen herrührt, und der Anwendungszustand, der von anderen Anwendungen abgeleitet wird. Die Inspektion auf Basis des Status erlaubt der Firewall von früheren Kommunikationsversuchen zu lernen und damit zukünftige Kommunikation besser zu bewerten. Obwohl Firewall-1 im 1. Quartal ‘97 veröffentlicht wurde, ist die Lernkurve des Produktes noch sehr flach und die Rate der Wiedererkennung noch nicht ausreichend, um einen wirklichen Schutz vor JAVA oder ActiveX-Komponenten zu bieten. In der Betrachtung der Sicherheit der beiden Komponentenmodelle muß also von einer „klassischen“ Firewall als Trennung zwischen dem Intra- und Internet ausgegangen werden.

2.4.3.3 Virtual Private Network (VPN)

Im Gegensatz zum topologisch fest gebundenen Einsatz der Firewalls, besteht eine Möglichkeit der Ausdehnung des Intranets über die örtlichen Grenzen des Unternehmens hinaus, in der Verwendung von sogenannten virtuellen, privaten Netzwerken. Besonders bei der Verbindung von Außenstellen über das offene, und damit unsichere Internet, sind VPN in der heutigen Phase der Globalisierung des Marktes ein aktuelles Thema, das nicht allein durch mit Firewalls geschützten LANs gelöst werden kann. VPN bieten auch eine Lösung für mobile Anwender und das sogenannte „mobile computing“.

Die Verwendung von Verschlüsselung für den Datenverkehr zwischen Firewalls, also über unsichere Transportwege, hat das Konzept der VPNs wieder aktuell werden lassen. Wenn zwei Intranets, also durch Firewalls geschützte TCP/IP Netzwerke, miteinander kommunizieren, muß deren Kommunikation geschützt werden. Gleiches gilt bei der Anbindung von Klienten in einem „Public Switched Telephone Network (PSTN)“ an einen Einwahl-, oder „Remote Access Server (RAS)“. Die Basis des VPN bildet der offene Standard des „Point-To-Point Tunnelling Protocol (PPTP)“. PPTP ist eine neue Netzwerktechnologie, die Multiprotokoll VPN unterstützt. Im Juni 1996 wurde auf der „Internet Engineering Task Force (IETF)“ Versammlung in Montreal das PPTP als Internet Draft Standard vorgestellt. Die technische Spezifikation von PPTP ist jetzt aktualisiert und als IETF Internet Draft öffentlich verfügbar [DRAFT962].

Ein VPN kann als die Fähigkeit, einen „Tunnel“ durch das Internet und andere öffentliche Netzwerke zu öffnen, beschrieben werden, so daß die Sicherheitsumgebung des Intranets durch diesen Tunnel auf andere Teilnetze ausgedehnt werden kann. Die Verbindung von Außenstellen eines Unternehmens über eine sichere PPTP Verbindung ist damit möglich. Aus Anwendersicht ist das zugrundeliegende physikalische Netzwerk irrelevant, da sich die Verbindung für ihn als dediziertes privates Netzwerk darstellt.

Ein VPN Tunnel verpackt auch Informationspakete, die nicht dem Internet-Standard für Adressierung entsprechen, in IP Pakete (siehe dazu Abb. 12). Nach dem Erhalt wird diese Verpackungsinformation wieder, für die höheren Schichten des OSI-Modells transparent, entfernt. Damit werden entfernte Anwendersysteme zu virtuellen Knoten des Netzwerkes, mit dem sie über das VPN verbunden sind. Das Intranet wird auf diese virtuellen Knoten, seien es Einzel- oder Mehrplatzsysteme, ausgedehnt.

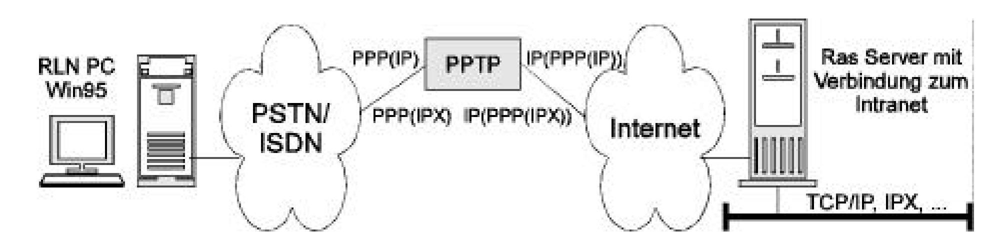


Abb. 12: Konzeptionelles Modell eines VPN.

Das konzeptionelle Modell eines VPN zeigt, wie IP-, IPX- oder NetBEUI basierte Intranets über das Internet verbunden werden können.

Das PPTP Protokoll selbst basiert auf „Point-to-Point Protocol (PPP)“[8] [TAN97] und TCP/IP. Dabei bietet PPTP Authentifizierung und Methoden zur Datensicherheit und -komprimierung. Darüber hinaus ermöglicht PPTP eine Sitzung über eine bestehende IP Verbindung zu übertragen, unabhängig davon wie diese Sitzung aufgebaut wurde. Eine Überbrückung von Firewalls ist dadurch kein Problem.

Medium	IP	GRE	PPP	PPP Nutzlast

Abb. 13: Verwendung von GRE im PPTP.

PPTP verwendet die „Generic Routing Encapsulation (GRE)“ bei dem Transport von PPP Paketen. Verschlüsselung wird auf die Nutzlast angewendet, ein Authentifizierungsprotokoll wird verwendet, um Objekte vor dem Zugriff zu authentifizieren [RFC 1701].

PPTP verwendet zwei Pakettypen:

1. *Datenpakete* enthalten die Benutzerdaten. Es sind PPP Pakete, die durch das GRE Protokoll V2 verkapselt sind.
2. *Kontrollpakete* werden für Statusabfragen und Signalisierung verwendet. Sie werden über TCP-Verbindungen übertragen und verwenden eine einzige TCP-Verbindung als Kontrollkanal.

Vor der Übertragung wird von den beiden Teilnehmern gegenseitig um die Erlaubnis für die Übermittlung der Pakete gebeten, und sowohl Kompressionsverfahren als auch Verschachtelungsmethode in einem Handshake ausgehandelt. Bei der Übertragung stellt das PPP Protokoll Serialisierung zur Verfügung.

[8] Die Protokollgruppe PPP ist im [RFC 1661] definiert und wird für Wählleitungen und Router/Router-Mietleitungen verwendet.

3 Verarbeitung mit verteilten Objekten

3.1 Einführung

Verteilte Verarbeitung ist ein Schlagwort der EDV in den 90er Jahren. Dieser Terminus wird stark mit dem Begriff *downsizing* assoziiert, der selbst als Schlagwort die Migration von großen EDV-Systemen, vom zentralen Mainframe zu Netzwerken von Workstations, Minicomputer und Personalcomputer beschreibt. Im Rahmen dieser Entwicklung wird den Internetcomputern eine große Zukunft eingeräumt (vgl. Kapitel 5.1).

Die ersten Versuche von verteilten Anwendungen beinhalteten Dateiserver, die es Anwendern ermöglichten, gegenseitig auf ihre Dateien zuzugreifen. Die Lokation der Datei war dabei bereits transparent. Als nächster Schritt wurden Datenbankanwendungen auf Netzwerken eingeführt, die ein zentrales Datenmanagement anboten und auf das Klient/Server-Modell basierten. LAN Produkte wie „Novell NetWare“ und „Microsoft LAN Manager“ integrierten Datei-, Druck- und Nachrichtendienste. Obwohl diese Modelle für ihre Aufgabe spezialisiert und damit effizient sind, liegt der Nachteil dieser „vertikalen“ Ansätze in ihrer mangelnden Fähigkeit für andere Zwecke adaptiert zu werden.

Im folgenden werden drei „horizontale“ Modelle (CORBA, OSF DCE und MS DCE) vorgestellt, und besonders der Begriff der Sicherheit auf diese Ansätze der verteilten, heterogenen Systeme ausgedehnt. Dabei werden die verschiedenen Modelle zur Interaktion von verteilten Objekten vorgestellt und verglichen.

3.2 Plattformen

3.2.1 CORBA

3.2.1.1 Allgemein

Die „Object Management Group (OMG)“ wurde im Mai 1989 als Interessenverband für Entwickler, Netzwerkbetreiber, Hardwareproduzenten und kommerzielle Anwender gegründet. Die Gründungsmitglieder sind 3Com, American Airlines, Canon, Data General, Hewlett-Packard, Philips, Telecommunications, SUN Microsystems und UniSys. Mit der „Object Management Architecture (OMA)“ wurde durch diesen Verbund eine Softwarearchitektur spezifiziert, welche die Kooperation von verschiedenen Anwendungen auf Basis verschiedener Computersysteme (Hard- und Software) ermöglicht und standardisiert. Ziel ist die Unabhängigkeit des Standards von Programmiersprachen, Betriebssystemen, Standorten und Hardware. Im Frühjahr 1996 zählte die OMG 600 Mitglieder.

Kern dieses Architekturmodells ist die Spezifikation des „Object Request Brokers (ORB)“ in der CORBA Version 1.1 von 1991. Er ist die von der OMA vorgesehene, universelle Middleware für beliebige Objekte auf heterogenen (verteilten) Systemen. Der entsprechende Standard wird als „Common ORB Architecture (CORBA)“ bezeichnet. CORBA isoliert den Klienten, und damit den Entwickler, von den verteilten, heterogenen Eigenschaften von Informationssystemen. Konzeptionell ist CORBA auf der Anwendungsschicht des OSI-Modells angesiedelt [MOV95].

Folgende Vorteile bietet CORBA:

- Der Standard spezifiziert eine Schnittstelle für Objekte, die vom Betriebssystem und Programmiersprachen unabhängig ist.
- Aufgrund des hohen Abstraktionsgrades muß der Entwickler sich nicht mit den niederen Schichten des OSI-Modells befassen, wie dies zum Beispiel bei der Socket Programmierung erforderlich ist (vgl. Kapitel 2.3.3).

- Der Entwickler muß sich nicht mit der Lokation des Servers oder dessen Zustand der Aktivierung befassen.
- Der Entwickler muß sich nicht mit der Hardware oder dem Betriebssystem befassen, auf dem der Server aktiv ist.
- Integrationsprobleme werden vereinfacht. So muß zum Beispiel bei der Übertragung von Informationen zwischen verschiedenen Systemen nicht auf die unterschiedlichen Darstellungen geachtet werden.

Im Rahmen dieser Arbeit sind ORBs als Basis für JAVA von besonderem Interesse. SUN Microsystems bietet ein solches Produkt unter der Bezeichnung „JAVA Objects Everywhere (JOE)" an, das die Kommunikation mit Applikationen ermöglicht, die auf Grundlage der eigenen CORBA Implementierung „Network Enterprise Objects (NEO)" entwickelt wurde. Konkurrenz erhält der Hersteller von Workstations von der Firma Visigenic, die mit „Black Widow" ebenso einen vollständig in JAVA programmierten ORB vorweisen kann, wie auch IONA mit „OrbixWeb". Letzteres ist in der Version 2.0 zudem in der Lage, jedes Endgerät mit Serverfunktionen zu versehen. Klientendaten können so von einer zentralen Applikation aus abgerufen werden [CW96].

3.2.1.2 Object Management Architecture (OMA)

Im Sinne der OMA werden *Objekte* als die grundlegenden Bausteine für verteilte Anwendungen definiert. Für ein Objekt können Operationen aufgerufen und Eigenschaften erfragt werden. Ein *Schnittstelle* (~ Interface) gibt Aufschluß über die Art der zur Verfügung gestellten Operationen und deren Übergabeparameter.

Der Hauptvorteil eines solchen, aus der objektorientierten Programmierung abgeleiteten Ansatzes, ist die Modularität, das heißt die Fähigkeit, durch Integration weiterer Objekte und durch Erweiterung vererbter Objekte, die Funktionsfähigkeit eines Objektes zu vergrößern. Diese Wiederverwendung von Objekten verspricht verkürzte Entwicklungszeiten, reduzierte Fehleranfälligkeit und bessere Amortisation von Entwicklungskosten [STR92].

Zentraler Aspekt der OMA ist die Beschreibung von Schnittstellen durch die „Interface Description Language (IDL)". Durch diese Sprache werden die öffentlichen Objekteigenschaften (Attribute, Operationen, Programmausnahmen und für die Kommunikation benötigte Datentypen) einheitlich spezifiziert. Um ein Objekt zu implementieren, werden gängige (objektorientierte) Programmiersprachen wie C++, JAVA, C usw. verwendet, für die jeweils eine Abbildungsvorschrift (*Language-Mapping*) die Umsetzung von, durch IDL beschriebene, Schnittstellen in Konstrukte der Programmiersprache definiert. Beachtenswert ist, das die IDL nicht die Implementierung des Objektes selbst beschreibt (ein konträrer Ansatz wird im Kapitel 3.2.3.2 vorgestellt). So können, je nach Mächtigkeit der eingesetzten Programmiersprache, Objekte Systemzugriffe durchführen, auf Datenbanken zugreifen oder Dienste unabhängig von der CORBA Spezifikation anbieten [RED96].

Die OMA unterteilt, im Unterschied zu COM(vgl. Kapitel 3.2.3.2), die Bestandteile einer Anwendungen in *Komponenten*, eine Menge von Objekten die gemeinsam eine Aufgabe erfüllen. Diese Aufgabe wird als *Dienst* bezeichnet.

3.2.1.3 Common Object Request Broker Architecture (CORBA) 2.0

Der aktuelle CORBA-Standard 2.0 liegt seit 1995 vor und besitzt folgende Eigenschaften [RED96]:

- Objektorientierung
- Verteilungstransparenz
- Effizienz
- Unabhängigkeit von Hardware, Betriebssystem und Programmiersprache
- Offenheit

CORBA ist der OMG Standard, der den Aufbau des ORB, seine Bestandteile sowie deren Verhalten und Schnittstellen beschreibt [OMG95]. Ein Klient kann unter Verwendung des ORB einem Objekt einen *Request* (~ Anfrage) zuschicken und von diesem, ebenfalls über den ORB, eine *Response* (~ Anwort) zurückerhalten. Ein Request wird von einem Objekt bearbeitet, indem das Objekt eine *Operation* ausführt und die übergebenen Argumente (Parameter) verarbeitet. Ein spezielles Ergebnis ist die *Exception*, ein sogenannter Ausnahmefehler, der den Klienten auf einen aufgetretenen Fehler bei der Bearbeitung seines Requests hinweist.

Ein *Klient* im Sinne des Klient/Server-Modells, ist eine Softwareeinheit, die von einem Objekt eine Operation anfordert. Die *Objektimplementierung* ist eine Menge von Programmcode, die das Verhalten eines Objektes beschreibt. Aus der Sicht des Betriebssystems werden mehrere Objektimplementierungen zu einem *Server* zusammengefaßt. Ein Server kann dabei mehrere Klienten besitzen.

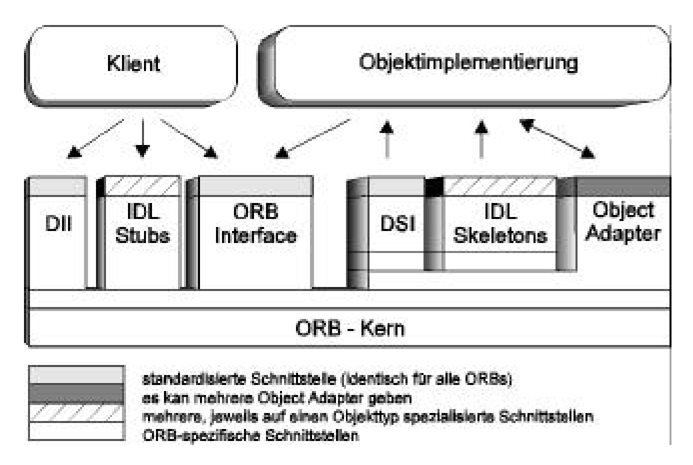


Abb. 14: CORBA-Modell mit ORB Schnittstellen [RED96].

Der ORB besteht aus DII, DSI, Objekt Adapter und ORB-Kern. Die *Stubs* und Skeletons werden je nach Bedarf vom Anwendungsprogrammierer aus den Schnittstellenbeschreibungen der verwendeten Objekte generiert und dem System hinzugefügt. Die Aufgaben von Stub und Skeleton verteilen sich analog zum RMI-Modell (vgl. Kapitel 2.3.5). Um einen Request zu initiieren, kann ein Klient entweder das „Dynamic Invocation Interface (DII)“ oder einen, vom IDL-Compiler für eine IDL-Schnittstellendefinition generierten, *Stub* benutzen. Im DII muß zur Laufzeit spezifiziert werden

1. welches Objekt angesprochen wird,
2. welche Operationen ausgeführt werden und
3. welche Argumente die Operation hat (Anzahl, Typ, Modus, Wert).

Die Verwendung von Stubs vereinfacht die Programmierung, da die Schnittstelle des aufgerufenen Objektes bereits in der Entwicklungsphase bekannt ist (dieses Verfahren wurde bei der Beispielimplementierung verwendet) [RED96].

Die Objektimplementierung empfängt einen Request, indem der ORB über ein *Skeleton* eine Methode aufruft. Das Skeleton ist speziell für einen Schnittstellentyp spezifiziert und wird vom IDL-Compiler aus der Beschreibung der Schnittstelle generiert. Als Alternative steht das „Dynamic Skeleton Interface (DSI)“ zur Verfügung, das beliebige Requests entgegennehmen und dem entsprechenden Skeleton zuweisen kann.

Der „Object Adapter (OA)“ setzt auf den ORB auf und ist für die Verbindung von Objektimplementierung an den ORB zuständig [MOV95]. OAs bieten Dienste wie Generation und Interpretation von Objektreferenzen, Methodenaufrufe, Objektaktivierung und -deaktivierung und die Abbildung von Objektreferenzen auf Implementierungen an. Verschiedene Typen von Objektimplementierungen haben unterschiedliche Anforderungen und müssen von unterschiedlichen OA,

zum Beispiel einem objektorientierten Datenbankadapter für Objekte innerhalb einer Datenbank, angebunden werden. Der „Basic Object Adapter (BOA)“ definiert einen OA, der für konventionelle Objektimplementierungen verwendet werden kann. CORBA spezifiziert dabei die Implementierung der ORB/BOA Funktionalität nicht [RED96].

CORBA verlangt zudem nicht, daß der ORB als eine einzige Komponente realisiert wird, was zum Beispiel beim Einsatz unter JAVA nicht möglich wäre. Statt dessen werden nur die Schnittstellen zum ORB definiert. Die von dem ORB angebotenen Operationen können damit in drei Gruppen unterteilt werden [RED96]:

- Operationen, die für alle ORBs identisch sind (allgemeine Schnittstellen zum ORB, DII und DSI).
- Operationen, die speziell für den Umgang bestimmter Objekte verwendet werden (Stubs und Skeletons).
- Operationen, die für einen bestimmten Stil von Objektimplementierungen typisch sind (OA).

Die entsprechenden Schnittstellen werden in Abb. 14 dargestellt.

3.2.1.4 OMG-Interface Description Language (IDL)

IDL stellt ein Mittel zur Beschreibung von Objektschnittstellen bereit. Diese Beschreibung erfolgt auf einem konzeptionellen Niveau, da keine der bekannten Programmiersprachen direkt die IDL-Definitionen umsetzen kann. Die in der IDL Syntax beschriebenen Schnittstellen müssen zunächst in äquivalente Konstrukte der betreffenden Programmiersprache übersetzt werden, so daß diese mit ihren Mitteln (Methodenaufrufe, Funktionsaufrufe, usw.) den Zugriff auf eine Schnittstelle realisieren und durchführen kann [MOV95].

Die Vorschriften für die Zuordnung sind in sogenannten *Language-Mappings* festgelegt, die für jede Programmiersprache spezifisch sind. Für C, C++ und Smalltalk sind die Language-Mappings heute standardisiert. Für JAVA [OMG97], Ada, Fortran und Objective-C sind entsprechende Standards der OMG vorgeschlagen[9].

Die Realisierung der im Language-Mapping beschriebenen Schnittstellen ist vom ORB abhängig. Sie kann, neben den durch CORBA standardisierten, auch proprietäre Schnittstellen zum ORB-Kern bereitstellen. Dabei wird versucht, die Details der Implementierung dem Entwickler transparent zu gestalten [RED96].

3.2.1.5 CORBA Services

Die *CORBA Services* sind Dienste der Systemebene, die IDL-spezifizierte Schnittstellen anbieten. Sie bereichern und vervollständigen die Funktionalität des ORB und werden verwendet, um Objekte zu erstellen, zu benennen und in die CORBA Umgebung zu integrieren. Die OMG Object Services bestehen aus der „Common Object Service Spezifikation (COSS)“, einer Serie aus mehreren Bänden, die jeweils aus einem „Object Services Request for Proposal (RFP)“ bestehen [MOV95]. Insgesamt sind 16 Services im CORBA 2.0 Standard spezifiziert [ORF97]:

RFP1 Services:

- Object Event Notification Service:
  Dynamische (De-)Registrierung der Schnittstellen aufgrund von Ereignissen.
- Object Lifecycle Service:
  Methoden, um Objekte auf dem Bus zu erstellen, kopieren, verschieben und zu löschen.
- Object Naming Service:
  Erlaubt Objekte anhand ihres Namens zu finden. Objekte können beim Naming Service (~ Namensdienst) angemeldet werden, so daß sie anhand ihres Namens lokalisiert werden können. Der Naming Service ermittelt die zu den Objekten gehörigen Objektreferenzen.

[9] Die CORBA Spezifikation mit Language-Mappings ist unter http://www.omg.org/corba/cichpter.htm zu recherchieren.

- Object Persistence Service:
  Eine Schnittstelle, um Objekte persistent auf verschiedenen Speicherservern zu sichern.

RFP2 Services:

- Object Concurrency Control Service:
  Sperrungsmanager zur Verwaltung von konkurrierenden Zugriffen auf Objekte.
- Object Externalization Service:
  Datenströme aus Objekten ein- und auslesen.
- Object Relationships Service:
  Dynamische Verbindung zwischen Komponenten, ohne daß diese die Spezifikation des Partnerobjekts kennen.
- Object Transaction Service:
  Zweiphasen Commit Transaktionskontrolle.

RFP3 Services:

- Object Security Service:
  Autorisations- und Überwachungsfunktionen werden auf Objektebene zur Verfügung gestellt.
- Object Time Service:
  Der Time Service stellt eine synchronisierte Zeit in einer verteilten Umgebung zur Verfügung.

RFP4 Services:

- Object Licensing Service:
  Lizenzierungsmechanismus für Verwendungsvergütung.
- Object Properties Service:
  Dieser Service erlaubt das dynamische Verknüpfen von Eigenschaften zu Objekten, unabhängig von ihrer Schnittstellendefinition.
- Object Query Service:
  Abfrageoperationen für Objekte auf Basis der „Structured Query Language (SQL)“ [ROD97], also Operationen auf Objektmengen, die eine prädikatenbasierte und deklarative Spezifikation besitzen. Abfragen können Objektmengen als Ergebnis zurückliefern.

RFP5 Services:

- Trader Service:
  Dieser „Makler“ ist eine Suchmaschine für IIOP Objekte, die komplexes Suchen für verteilte Objektdienste ermöglicht.
- Collection Service
  Unterstützung für die Gruppierung von Objekten und deren Manipulation als Gruppe.
- (Start-up Service)

Damit greift COSS nahezu alle Aspekte der verteilten Verarbeitung auf. Im Unterschied zu der COM Spezifikation von Microsoft unterscheidet sich aber der Umfang des Standards CORBA 2.0 stark von seiner tatsächlichen Realisierung. Um den aktuellen Stand der verfügbaren CORBA Services darstellen zu können, wird hier exemplarisch die Produktlinie des Marktführers für ORBs, die Firma IONA, vorgestellt[10].

OrbixEvent ist die Implementierung des Ereignisdienstes als OrbixEvent Serveranwendung. Diese Anwendung bietet eine IDL-Schnittstelle, die drei verschiedene Objekte spezifiziert: Anbieter, Konsumenten und Ereigniskanäle. OrbixEvent ist IIOP fähig, unterstützt als Orbix 2.x Multi-Thread-basierte Anwendung sowohl Single- als auch Multi-Threaded Klienten und ist auf den Betriebssystemen Solaris und WinNT verfügbar. Der Naming Service von IONA wird durch *OrbixNames* vollständig implementiert. OrbixNames V1.0.3 ist kompatibel zu Orbix 2.1c, Orbix 2.2 und OrbixWeb, und auf den Betriebssystemen Solaris 2.x, WinNT 4.0, HP 10.10 und AIX

[10] Einen aktuellen Stand der implementierten Dienste findet sich unter http://www.iona.com/Products/Services/index.html.

4.1 verfügbar. Mehrfache und verschachtelte Kontexte werden unterstützt. *OrbixTrader* ist die Implementierung des OMG Trading Object Services durch IONA, welcher eine verbesserte Ressourcenlokalisierung bei Klienten-Server-Interaktionen in VS bietet. Der OrbixTrader ist auf Solaris verfügbar. Versionen für HP und WinNT waren für den Oktober 1997 angekündigt. Weiter wird der „Object Transaction Service (OTS)“ durch *OrbixOTS* realisiert. Dadurch wird die einfache Entwicklung von Anwendungen ermöglicht, die Transaktionen und verteilte Datenintegrität erfordern. OTS unterstützt dabei eine verteilte 2-Phasen Commit ACID[11]-Transaktion [RIC90]. Die Transaktionsanforderung durch den Klienten (z.B. auch JAVA-Anwendungen) wird vom OTS-Transaktionsmanager transparent verarbeitet. Es werden zwei OTS Lösungen angeboten: OrbixOTS der Firma TRANSARC auf Basis des ENCINA TP Subsystems bietet eine in der Industrie erprobte Lösung für verteilte OTS-Transaktionen, und OrbTP von Groupe Bull eine neue Implementierung der OTS-Spezifikation. Beide Lösungen stehen als Betaversion zur Verfügung und waren für Mitte 1997 angekündigt. Der CORBA Sicherheitsdienst bietet zwei Ebenen von Informationssicherheit. Die erste behandelt die fundamentalen Strukturen für die Konstruktion von sicheren VS. Die zweite stellt eine Menge von Schnittstellen für die Administration von Sicherheitsregeln und -domänen zur Verfügung. Nachdem die OMG die Wichtigkeit von SSL-Sicherheit (vgl. Kapitel 2.3.4) erkannt hat, wird an einer standardisierten Einbindung in ORBs gearbeitet. IONA bietet zur Zeit eine Implementierung der ersten Ebene als Betaversion auf Solaris 2.x und WinNT V3.51 an. Diese für das 3. Quartal ‘97 angekündigten Versionen verwenden die DCE/Kerberos Sicherungskomponente (vgl. Kapitel 3.2.2.5) um die „Generic Security Service (GSS)“-API-Funktionalität zu bieten [RFC 1508]. Die Integration von SSL wird als Teil des ORBIX/„Object Transaction Monitor (OTM)“ [IONA297] als Betaversion ab dem 3. Quartal 1997 angeboten. Abschließend bietet der OrbixManager Konfigurationsmöglichkeiten und Performanzmanagement für Orbix und OrbixTalk. Der Orbix-Manager verwaltet die CORBA-basierten Middlewarekomponenten einer Anwendung. Er besteht aus einer Vielzahl von Einzelkomponenten wie GUI, Systemwerkzeugen und verschiedenen Monitoren.

Die folgende Tabelle[12] stellt den Stand der Realisierung dieser Produktlinie dar.

CORBA 2.0	***ORBIX Service (realisiert)***	***Plattform***			
		WinNT	***HP***	***Solaris***	***AIX***
Live Cycle					
Persistence					
Naming	OrbixNames	x	x	x	x
Event	OrbixTalk	x	x	x	x
Concurrency					
Transaction	OrbixOTS	3. Q. ‘97	3. Q. ‘97	3. Q. ‘97	3. Q. ‘97
Relationship					
Externalisation					
Query					
Licensing					
Properties					
Time					
Security	OrbixSecurity	3. Q. ‘97		3. Q. ‘97	
Trader	OrbixTrader	3. Q. ‘97	3. Q. ‘97	x	
Collection					
Start-up					

Tab. 9: Implementierungen der CORBA 2.0 Spezifikation durch IONA.

[11] „Atomicity, Consistency, Isolation and Durability (ACID)“ [MSSQL96].

[12] Nach Auskunft von Geoff Swain, Unix A/C Manager, IONA (mailto:gswaine@iona.com).

Es wird deutlich, daß selbst der Marktführer nur einen Bruchteil der CORBA 2.0 Spezifikation realisiert hat. Genau hier greift aber ein Kritikpunkt an, der den CORBA-Standard als bloße Theorie bezeichnet [MS952]. Eine Vielzahl von Implementierungen und Anwendungen auf Basis von CORBA entkräften diesen Vorwurf zum Teil, ein Markt für wiederverwendbare CORBA-Objekte existiert aber nicht.

3.2.1.6 Internet Inter-ORB Protocol (IIOP)

Für die schnelle, bidirektionale Verbindung von CORBA-kompatiblen Objekten erhob die „Object Management Group (OMG)“ das „Internet Inter-ORB Protocol (IIOP)“ in CORBA 2.0 zum Standard. Die Firma Netscape wird dieses Protokoll voraussichtlich schon in der nächsten Version des Browsers namens „Netscape Communicator V4.0“ und in den Serverprodukten des Suitespot-Pakets unterstützen. Zusätzlich will die Firma den ORB von Visigenic in ihre nächste Produktgeneration einbauen, die Anfang 1998 auf den Markt kommen soll. Sie wird damit auch anderen Herstellern objektbasierte Schnittstellen anbieten. Hewlett-Packard will diese nutzen und Netscape Server über IIOP an das System Management Produkt „OpenView“ anbinden [CW96]. Aufgabe des IIOP ist vollständige Transparenz der Lokalität von Objekten bei Verteilung. Es gibt zwei verschiedene Möglichkeiten der Kooperation zwischen ORBs:

1. Alle ORBs unterstützen ein einheitliches Protokoll zwischen dem ORB auf Seite des Klienten und dem des Servers. Die Spezifikation eines solchen Protokolls wurde von der OMG im „General Inter-ORB Protocol (GIOP)“ festgelegt.
2. ORBs unterstützen oft proprietäre Protokolle, die aber über Brücken für die Protokollumwandlung verbunden werden können.

Das IIOP ist eine Spezialform des GIOP, welches auf der Protokollfamilie TCP/IP basiert und im CORBA 2.0 Standard detailliert beschrieben ist. Für spezielle Dienste sind angepaßte Protokolle, sogenannte „Environment-Specific Inter-ORB Protocols (ESIOP)“ vorgesehen. Ein Beispiel für ein ESIOP ist das auf OSF DCE (vgl. Kapitel 3.2.2) basierende „DCE Common Inter-ORB Protocol (DCE-CIOP)“ [RED96].

Bei Brücken zwischen ORBs werden Vollbrücken, die direkt das Protokoll des einen ORB in das des anderen umsetzen, und Halbbrücken, die das Protokoll des einen ORB zunächst in ein allgemeines Protokoll (z.B. IIOP) und dann wieder in das spezifische Protokoll des zweiten ORB überführen, unterschieden. Hardware Brücken (wie zum Beispiel eine Firewall oder ein „*Bastion Host*“ [BRU97]) von Netzwerksegmenten sind ein naheliegender Ort für die Implementierung von ORB-Brücken.

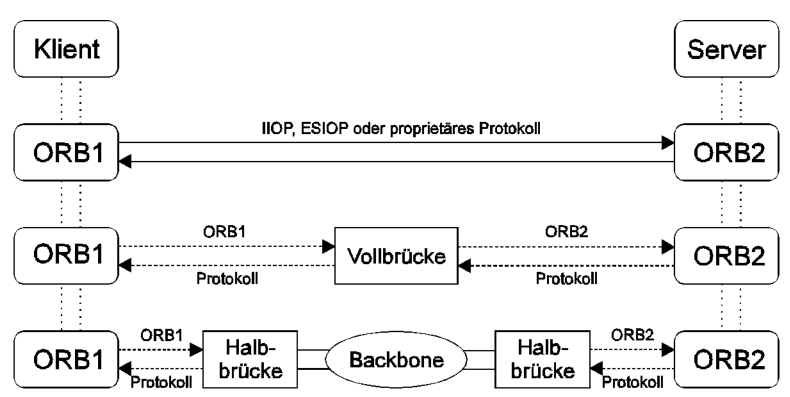


Abb. 15: Kommunikationsmodelle zwischen ORBs [RED96].

Die grundlegende Kommunikation über das IIOP stellt sich wie folgt dar: Wenn ein IIOP-Klient eine Nachricht an ein entferntes Objekt sendet, ist dafür eine „Interoperable Object Reference (IOR)“ erforderlich, welche die Adressierungsinformation für das Objekt speichert. IORs sind die ORB-übergreifende gültige Form der Objektreferenz.

Eine IOR besteht typischerweise mindestens aus [IONA197]:

- dem Namen des Servers, auf dem das Objekt vorhanden ist
- dem Port, auf dem das Objekt erreichbar ist
- dem Objektschlüssel (ein Bytestring, der das Objekt identifiziert)

Sobald ein IIOP-Klient die IOR des entfernten Objektes erhalten hat, wird eine bidirektionale TCP-Verbindung aufgebaut. Diese Verbindung kann auch für die Kommunikation mit anderen Objekten auf demselben Host benutzt werden. Das IIOP-Modell basiert dabei auf zwei Nachrichtentypen: Der Klient sendet einen *Request*, auf den der Server mit einem *Reply* reagiert. Weitere Nachrichten behandeln Time-outs (~ Zeitüberschreitungen) und unerwartete Fehler (Exceptions).

Um dieses Protokoll durch eine Firewall zur führen sind zusätzliche Produkte, wie zum Beispiel ORBIX WonderWall der Firma IONA notwendig. Analog zu paketfilternden Gateways, analysieren und filtern solche Produkte eingehende Requests auf Basis folgender Informationen aus dem Nachrichtenkopf [IONA197]:

- Nachrichtentyp
- IP-Adresse des Klienten
- Objektschlüssel des angeforderten Objekts
- Name der aufgerufenen Methode
- Dem Principal, der den Klienten aufgerufen hat
- Beliebigen IOP Service Kontexten

Der Körper der Nachricht selbst kann ohne Kenntnis der IDL nicht gefiltert werden.

3.2.2 Distributed Computing Environment (DCE)

3.2.2.1 Einführung

Wie bereits verdeutlicht wurde, bildet das Komponentenmodell COM der Firma Microsoft die einzige Konkurrenz zum CORBA-Modell beim Einsatz über das Internet. Microsoft hat sein Modell stark an das „Distributed Computing Environment (DCE)“ angelehnt. Eine genaue Untersuchung dieses Modells bietet sich also an und stellt die Grundlage für die sich anschließenden Besprechung von WinNT als Plattform für COM dar. Dabei müssen dann nur noch die Unterschiede zu DCE detailliert erläutert werden. In der Strukturierung der Kapitel und Absätze, wurde darauf geachtet, daß inhaltlich auch Parallelen zur Diskussion des CORBA-Standards aufgezeigt werden können.

Die „Open Software Foundation (OSF)“ ist ein Konsortium von Firmen, die in Zusammenarbeit Software für den Markt der offenen Systeme entwickeln. Die OSF entscheidet, welche Technologien zu implementieren sind, und lädt dann Mitglieder des Konsortiums ein, um diese Software unter Leitung der OSF zu entwickeln. Innerhalb der OSF wird der Quellcode dieser Entwicklung dann wiederum an alle Mitglieder weitergegeben und so dessen Entwicklung gefördert. Die OSF DCE Software wurde so durch Beiträge von Digital Equipment Corporation DEC (CDS, DTS, Teile des RPC und Threads), Hewlett-Packard Company (Teile des RPC und Sicherheit), International Business Machines Corporation IBM (Softwareintegration), Siemens Nixdorf Informationssysteme AG (GDS, XDS) und Transarc Corporation (FDS und LFS) entwickelt [ROS931].

DCE bietet eine Menge von integrierten Diensten (Bibliotheken, Werkzeugen und Laufzeitdiensten) an, um ein VS zur Verfügung zu stellen und die Unterschiede zwischen verschiedenen Hosts transparent zu gestalten. Die Dienste arbeiten über mehrere Systeme und bleiben von einzelnen Systemen unabhängig. DCE Dienste sind vor einem „single point of failure“ durch Replikation der Dienste und wichtiger Komponenten auf entsprechenden Hosts gesichert. Diese Replikation sichert die Verfügbarkeit von kritischen Informationen auch bei Teilausfällen des

Systems. Trotz starker Interdependenzen zwischen DCE Komponenten, erlaubt eine dezentrale Kontrolle jede Komponente unabhängig von anderen zu administrieren. Dadurch kann DCE über LAN und WAN ausgedehnt werden und so bei Bedarf mehrere tausend Systeme bedienen.

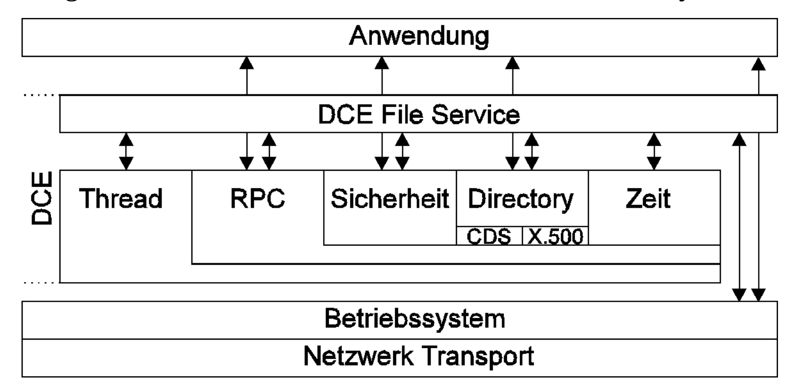


Abb. 16: DCE Konzeptmodell [ROS931].

Das DCE Konzeptmodell verdeutlicht, daß die DCE-API eine Softwareschicht zwischen Anwendung und Betriebssystem schiebt, welche die Unterschiede der einzelnen Hostsysteme verbirgt. RPC- und Thread-Dienste sind dabei Grunddienste, auf welche die anderen DCE Dienste aufbauen. Es wird weiter deutlich, wie Anwendungen die DCE-API verwenden können. Dabei werden nur die Dienste in Anspruch genommen, die auch tatsächlich benötigt werden. In der obigen Darstellung werden beispielhaft die Dienste RPC, Sicherheit, Verzeichnis und natürlich das Betriebssystem selbst verwendet [ROS931] und dies durch Pfeile zur Anwendungsebene dargestellt.

DCE bietet keine direkte Unterstützung für OOP; RPCs bieten aber einige unterstützende Merkmale für objektorientierte Systeme. Zum Beispiel kann ein RPC direkt an ein Objekt versandt werden. Die in DCE verwendeten RPCs sind inhärent synchron. Daher unterstützt DCE „Message Queuing“ (vgl. Kapitel 2.2.5), um eine asynchrone Verarbeitung zu ermöglichen. Auch Transaktionskontrolle wird semantisch nicht direkt von RPCs unterstützt, da eine individuelle Verwaltung einer Gruppe von Nachrichten nicht vorgesehen ist, eine Transaktionssemantik kann aber in DCE Anwendungen kodiert werden.

DCE 1.0 folgt verschiedenen offenen Standards, wie dem „Berkeley Internet Name Domain (BIND)“ und dem „International Telegraph and Telephone Consultative Committee (CCITT)“ X.500 Verzeichnisdienst [ROS931].

3.2.2.2 Zellen

Um die Administration von großen DCE Umgebungen zu vereinfachen, teilt DCE das Gesamtsystem und Teilsysteme in sogenannte Zellen auf. Jede dieser Zellen stellt die DCE Dienste, die für ein VS benötigt werden, zur Verfügung. Diese Dezentralisation minimiert „bottlenecks“ (Engpässe) [ROS931]. Die Last wird auf mehrere Hosts verteilt anstatt sie auf einen einzelnen, zentralen Server zu konzentrieren. Dadurch wird auch eine stärkere Flexibilität des Systems erreicht.

> *Eine **Zelle** ist das Grundelement für Operation und Administration in DCE. Eine Zelle ist also eine Gruppe von Anwendern, Systemen und Ressourcen, die normalerweise einen gemeinsamen Zweck haben und gleiche DCE Dienste verwenden.*

Die Grenzen einer Zelle, in Bezug auf Anzahl von Anwendern und Systemen, werden durch vier Punkte beeinflußt [ROS931]:

1. Da das Konzept der Zellen auf geteilte Ressourcen und DCE Dienste basiert, ist es sinnvoll, die Grenze einer Zelle durch den Anwenderkreis zu bestimmen, der ein gemeinsames Ziel

unter Verwendung der gleichen Ressourcen verfolgt. Geographische Überlegungen können, müssen aber keinen Einfluß auf diese Grenzen haben.

2. Eine Zelle sollte mindestens aus einem Verzeichnisserver, einem Sicherheitsserver und drei Zeitservern bestehen. Die Verfügbarkeit von qualifiziertem Personal zur Administration dieses Systems gibt Aufschluß auf die mögliche Größe einer Zelle.

3. Alle Objekte in einer Zelle müssen sich vor Durchführung einer Aufgabe authentifizieren. Mitglieder einer Zelle sollten sich gegenseitig und der Zelle selbst vertrauen können. Schäden an der Zelle sind mit wachsender Größe schwerer zu beheben. Abschließend müssen kooperierende Zellen sich ebenfalls gegenseitig authentifizieren, was bei n kooperierenden Zellen zu zusätzlichen $n(n-1)$ Paßwörtern führt. Zellen müssen also klein genug sein, um wartbar und vertrauenswürdig zu bleiben, müssen aber ausreichend groß sein, um für die Kooperation keinen zu großen Overhead zu verursachen.

4. Daraus resultierend, verursachen Namenauflösung und Authentifizierung mehr Last auf dem Netzwerk bei zellenübergreifenden Übertragungen, als bei zelleninterner Kommunikation. Die Zellengrenze wird damit auch von den Ressourcentypen einer Zelle und von der Häufigkeit des Zugriffs auf diese Ressourcen durch die Anwender einer Zelle determiniert.

Die Abbildung eines LAN-Segmentes auf eine Zelle ist nicht erforderlich, aber möglich. Die Segmentierung erfolgt eher durch gemeinsame Ziele und aus Überlegungen der Lastverteilung. In der praktischen Anwendung werden DCE-Zellen aber oft durch LAN-Segmente charakterisiert.

3.2.2.3 DCE-Interface Definition Language (IDL)

In monolithischen Anwendungen können direkte Prozeduraufrufe auf vordefinierte und statische Prozeduren innerhalb der Anwendung erfolgen. Die Prozeduren interpretieren die übergebenen Daten korrekt, da durch den gemeinsamen Adreßraum alle Definitionen zur Verfügung stehen. In verteilten Systemen liegt diese homogene Umgebung nicht vor, und Klienten und Server müssen sich über die Definition von Datentypen und Deklarationen von Prozeduren verständigen. Weiter wird eine Methode benötigt, um systembedingte Unterschiede (z.B. in der Länge von Variablen) zu maskieren.

Diese Übereinkunft wird, wie auch in der CORBA Spezifikation, über IDL ermöglicht. Beide Seiten verwenden diesen „Schnittstellenvertrag“, um gegenseitig Daten vernünftig interpretieren zu können. Um eine Schnittstellendefinition eindeutig zu gestalten, ermöglicht das Werkzeug *uuidgen* die Generierung von global eindeutigen Bezeichnern.

Obwohl der Sprachumfang und die Syntax hier nicht näher besprochen werden (siehe dazu [ROS932]), soll auf den wichtigen Punkt der Zeigerübergabe eingegangen werden, da sich hier ein Hauptunterschied zu WinNT (und damit zu COM) aufzeigen wird. Bei RPCs liegen aufrufender Prozeß und die aufgerufene Prozedur in unterschiedlichen Adreßräumen. Dadurch müssen Zeiger mit den durch sie referenzierten Daten (Ziel) an den Aufgerufenen übergeben werden. Dieser Vorgang wird für den Entwickler transparent, da ein Zeiger auf eine komplexe Datenstruktur als Parameter in einem RPC übergeben wird. Der Vorgang des Marshalling ist transparent. Um dies zu ermöglichen, muß die referenzierte Datenstruktur in der IDL Datei definiert werden.

Hierfür stell die OSF DCE Architektur drei verschiedene Zeigertypen zur Auswahl [ROS931]:

1. Reference Pointer
   Referenzzeiger werden bei einfachen Verweisen verwendet. Dabei muß sichergestellt sein, daß diese nie auf NIL verweisen. Diese einfachste Form eines Zeiger adressiert lediglich Daten. Der Stub Code bleibt klein und gestaltet diese Konfiguration sehr performant. Der Verweis auf einen Zeiger ist nicht möglich.

2. Unique Pointer
   Wenn ein Zeiger auch auf NIL verweisen muß, muß ein *eindeutiger Zeiger* verwendet werden. Der Verweis auf Zeiger und kaskadierende Zeiger ist auch hier nicht vorgesehen. Eindeutige Zeiger müssen ebenfalls vor dem Gebrauch initialisiert werden. Der Stub Code wird in diesem Fall um eine Fehlerbehandlung für Speicherschutzverletzungen erweitert. Leider werden eindeutige Zeiger von vielen DCE Implementierungen nicht unterstützt

3. Full Pointer
   Volle Zeiger decken den Bedarf für Referenzen auf komplexen Datenstrukturen mit Schleifen (z.B.: doppelt verkettete Listen) und Alias-Zeigern. Sie werden auch dann verwendet, wenn ein Zeiger auf NIL erforderlich ist und von der DCE Umgebung keine eindeutigen Zeiger unterstützt werden. Hierbei muß der Stub Code auch Endlosschleifen in Zeigerfolgen abdecken. Dies wird dadurch gelöst, daß jedem vollem Zeiger eine eindeutige Knotennummer zugeteilt wird.

3.2.2.4 Remote Procedure Call (RPC)

Der RPC ist eine verteilte Programmiermethode, bei der eine Anwendung auf einem System (Klient) eine Prozedur auf einem anderen, entfernteren System (Server) aufrufen kann. Dabei werden die Unterschiede zwischen heterogenen Systemen für den Entwickler transparent, da die Datenkonvertierung vom RPC System übernommen wird.

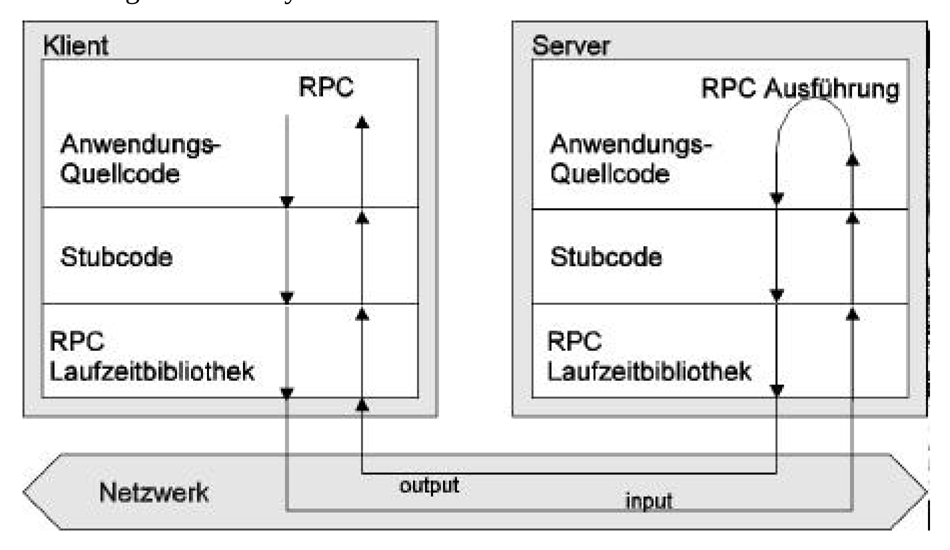


Abb. 17: RPC-Modell.

Bei verteilten Adreßräumen stellt sich das Problem der Speicherfreigabe. DCE Klienten verwenden dafür die standardisierten C Funktionen *malloc* und *free*. Auf der Serverseite stellt OSF DCE eine spezielle Stub Unterstützungsroutine zur Verfügung, um Speicher zu allokieren und freizugeben. Speicher, der auf der Serverseite durch den Aufruf von `rpc_ss_allocate()` allokiert wurde, wird nach Rückgabe der Parameter an den aufrufenden Klienten automatisch freigegeben.

Besonders in der potentiell instabilen Umgebung des Internets ist die Freigabe von stillstehenden Klientenressourcen ein abschließend wichtiger Punkt. OSF DCE-Server veröffentlichen eine asynchrone *ShutDown* Nachricht an einen Klienten, von dem seit 20 Sekunden keine Anforderung erfolgt ist. Damit wird der Klient aufgefordert die durch den RPC belegten Ressourcen freizugeben.

3.2.2.5 DCE Services

Der Sicherheitsdienst von DCE ermöglicht die Überprüfung der Identität eines Prozesses auf einem anderem System und der darauf basierenden selektiven Zuweisung von Ressourcen. DCE bietet diesen Dienst an, da von dem unterliegenden Betriebssystem im allgemeinen keine Unterstützung bei verteilter Sicherheit erwartet werden kann[13].

Der Sicherheitsdienst von OSF DCE beantwortet viele Fragen, wie zum Beispiel Schutz von Paßwörtern und Spoofing, die mit der Anwendung von TCP/IP entstehen. DCE setzt im Gegensatz zu CORBA explizit voraus, daß das Netzwerk unsicher ist und das jede übertragene Information potentiell abgehört und modifiziert werden kann. Vertrauen wird in ein nicht authentifiziertes Objekt auf Klienten- oder Serverseite nicht gesetzt.

Der DCE Sicherheitsdienst bietet [BRU97]:

- einen Registry-Service für Klienten und Server pro Zelle
- sichere Authentifizierung von Klienten und Server
- Nachrichtensicherheit, -integrität und -geheimhaltung über nicht vertrauenswürdige Netzwerke
- ACLs die es Servern ermöglichen, autorisierten Zugriff auf ihre Ressourcen zu gestatten
- Sicherheitsdienste und Administration sind von denen anderer Zellen isoliert

Der Authentifizierungsmechanismus von DCE basiert auf dem Kerberos[14] Authentifizierungsmodell [BRU97], welches im „Massachusetts Institute of Technology (MIT)“ in den späten 80er Jahren entwickelt wurde und 1993 als „Request for Comment (RFC)“ 1510 [RFC 1510] spezifiziert wurde. Die Grundannahme des Modells besteht darin, daß dem zugrundeliegenden Netzwerk nicht vertraut werden kann. Es verwendet eine Kombination von geheimen Schlüsseln um wechselseitige Authentifizierung von Klienten und Server zu ermöglichen.

Die DCE Authentifizierung basiert auf folgenden Einzelannahmen [BRU97]:

- Übertragene Informationspakete werden, falls nicht verschlüsselt, abgehört. Im Klartext übertragene Paßwörter können erkannt und „geheime“ Paßwörter entdeckt werden.
- Informationspakete, die Authentifizierungsinformationen tragen, können abgefangen, untersucht und zu einem späteren Zeitpunkt wiederholt werden.
- Netzwerkadressen kann nicht vertraut werden (weder IP- noch MAC-Adressen).
- Sowohl Klienten als auch Server müssen authentifiziert werden.

Der Prozeß der Authentifizierung verwendet eine Kombination von Paketverschlüsselung, zeitbasierten Beglaubigungen und einem vertrauenswürdigem „Dritten“, einem Dienst der sichere Authentifizierung bietet. In der DCE Terminologie wird synonym zum Objekt der Terminus des *Principals* verwendet. Dieser Begriff bezeichnet jedes Objekt (Person, Anwendung, Computer, Dienst oder Zelle), dessen Identität überprüft werden muß. Die Beglaubigung (Credential) ist ein elektronisches Zertifikat, das bei Darlegung die Identität des Principals belegt. In dem in dieser Arbeit vorgestellten Sicherungsmodell entsprechen Objekt und Subjekt dieser Aufteilung. Paßwörter und Konversationsschlüssel werden verschlüsselt, nicht aber die eigentlichen Transportdaten. Ein Konversationsschlüssel ist eine vom System generierte Zufallszahl, die zum Schutz der kommunizierenden Parteien eingesetzt wird. Wenn authentifiziert, werden Objekten zeitsensitive Beglaubigungen zugeteilt, die der zerstörende Wiederholung von Informationen widerstehen können.

Der DCE Sicherheitsdienst unterteilt sich in drei Komponentendienste [BRU97]:

1. Authentifizierung
   Der Authentifizierungsdienst stellt einem Principal, nach Erhalt eines gültigen Kennwortes,

---

[13] Im weiteren wird dargestellt werden, daß dies ins besondere bei MS WinNT der Fall ist und sich COM auch auf diese Tatsache in großem Maße stützt.

[14] Kerberos ist eine Gestalt der griechischen Mythologie, welche die schwerste Prüfung an Herkules verkörperte.

eine Berechtigung, die als „Ticket Granting Ticket (TGT)“ bezeichnet wird, zur Verfügung. Damit überprüft dieser Dienst die Identität eines Objektes beim ersten Kontakt mit dem System.

2. Privilegierung
   Dieser Dienst fügt sogenannte „Extended Privilege Attribute Certificates (EPAC)“, also zusätzliche Sicherheitsattribute, an die Berechtigung an.
3. Registrierung
   Der Registrierungsdienst ist eine Sammlung von Informationen über die Principals, Gruppen und Konten innerhalb der DCE-Zelle.

Jede Ressource (Person, Datei, Gerät usw.) innerhalb einer Zelle hat einen Bezeichner, der im Raum der verbundenen Zellen eindeutig ist. Der DCE Directory Service speichert die Bezeichner von Ressourcen in DCE. Dabei wird ein Bezeichner auf die eindeutige Netzwerkadresse der Ressource abgebildet. Die Komponente, die dabei die Namen innerhalb einer Zelle verwaltet, wird als „Cell Directory Service (CDS)“ bezeichnet. Demgegenüber werden die Namen außerhalb einer Zelle durch den „Global Directory Service (GDS)“ verwaltet. GDS ist eine Implementierung des Verzeichnisdienst-Standards X.500 [HAL96] der sowohl in der CCITT X.500 Reihe als auch in dem ISO 9594 Standard der „Intentional Standards Organisation (ISO)“ spezifiziert ist [ROS931].

DCE Klienten- und Serveranwendungen interagieren mit dem DCE Directory Service über einen Vermittler, der als *CDS Clerk* bezeichnet wird. Dieser übernimmt die Aufgabe der direkten Kommunikation der einzelnen Anwendungen mit dem Directory Service und erleichtert so die Implementierung erheblich.

Der „Domain Name Service (DNS)“ ist ein weitverbreiteter, globaler Namensdienst, der heute in vielen Netzwerken als Namensdienst für Internet-Hostnamen verwendet wird. Dieses Konzept wird auch unter WinNT und NIF von SUN Microsystems aufgegriffen. Die Spezifikationen des DNS sind in den [RFC 1034] und [RFC 1035] festgelegt.

Der verteilte Dateidienst „Distributed File Service (DFS)“ veröffentlicht eine einzelne Verzeichnisstruktur über alle Dateisysteme einer Zelle. DFS findet dann seine Bedeutung für Anwender, wenn diese immer einen eindeutigen Bezeichner für eine Datei angeben möchten, unabhängig davon, von welcher Lokation sich der Anwender an das System angemeldet hat. Für die Programmierung selbst ist dieser Dienst kaum von Bedeutung, da in VS auf Basis von Klient/Server-Modellen die Daten fast immer auf eindeutigen Servern innerhalb des Systems abgelegt sind [ROS931].

Der „Distributed Time Service (DTS)“ synchronisiert alle Systeme einer Zelle. Prozesse auf unterschiedlichen Systemen können die aktuelle Zeit vom DTS abfragen und eine homogene Antwort erwarten. Dieser Dienst wird vor allem bei spezialisierten Anwendungen mit Echtzeit-Beschränkung, aber auch bei „paralleler“ Verarbeitung und konkurrierendem Zugriff auf Informationen in VS verwendet.

3.2.2.6 Threads

Threads sind die Grundentität auf deren Basis das Betriebssystem Rechenzeit verteilt wird. Ein Thread kann jeden Teil des Anwendungscodes, einschließlich dem, der zur Zeit durch einen anderen Thread ausgeführt wird, ausführen. Alle Threads eines Prozesses teilen sich den gleichen virtuellen Adreßraum und die Ressourcen des Betriebssystems für diesen Prozeß [MS942]. Diese Technologie wird zunehmend eingesetzt, da sie die Performanz eines Anwendungssystems erheblich steigern kann. Programmierer können Bibliotheken von Threads ineinander verschachteln, um so ausgesetzte mit aktiven Operationen zu verflechten und damit die Auslastung des Prozessors auf einem relativ konstanten Niveau zu halten. Die Implementierung von DCE-Threads basiert auf den POSIX 1003.5a (Draft 4) Standard. Ihr Einsatz im DCE Umfeld liegt

vor allem bei Servern, die damit mehrfache, simultane Anforderungen von verschiedenen Klienten abarbeiten können.

3.2.3 COM und MS Windows NT

3.2.3.1 WinNT als Plattform für COM

Nach einer ausführlichen Behandlung der DCE-Spezifikation sollen im folgenden vor allem die Unterschiede zwischen OSF DCE und dessen Pendant in der Implementierung des Betriebssystems WinNT der Firma Microsoft aufgezeigt werden. Letzteres soll im weiteren als *MS DCE* bezeichnet werden, da eine einheitliche Namensgebung auch durch die Firma Microsoft nicht vorliegt und die vielen Ähnlichkeiten diese Benennung unterstützen. Durch ein besonderes Augenmerk auf die plattformübergreifende Interoperabilität zwischen den Systemen kann dann auf eine weiter Differenzierung zu OSF DCE verzichtet werden. Nachdem die Plattformen für Middleware behandelt sind, kann anschließend ausführlich auf die technischen Details des COM Modells eingegangen werden, das als neues System in der Öffentlichkeit noch weitgehend unbekannt ist.

WinNT ist ein Betriebssystem, welches sowohl auf „high-end“ Personalcomputer als auch auf Workstation-Architekturen (auf Basis von INTEL, PPC, Alpha und DEC) lauffähig ist. Dabei deckt WinNT alle Bereiche des Betriebssystems, über die grafische Benutzerschnittstelle bis hin zur Netzwerkkommunikation, ab. Teil von WinNT ist ein RPC-Mechanismus, der auf Protokollebene mit OSF DCE kompatibel ist. Dieser kleine Verbindungspunkt hat eine große Bedeutung, da er die Unterschiede der Architekturen transparent macht. Damit ermöglicht er [ROS932]:

- Die Kommunikation von DCE-Servern mit WinNT Klienten
- Die Kommunikation von WinNT Servern mit DCE Klienten
- Die Kommunikation von DCE-Servern mit Klienten auf Basis von MS-DOS

3.2.3.1.1 WinNT Domänen

Im Gegensatz zum Internet ist, im Umfeld von WinNT, eine Domäne eine Menge von Computern die eine gemeinsame, zentralisierte Sicherheitsdatenbank verwenden. Diese administriert den Zugriff auf die Domäne und implementiert Sicherheitsregeln. Auf dieser Ebene entsprechen die Domänen den OSF DCE-Zellen. Letztere können in der Praxis allerdings weit größere Ausmaße annehmen, als ihre WinNT Pendants. Die Domäne wird durch einen eindeutigen Bezeichner identifiziert und die Menge der enthaltenen Computer bildet eine verteilte Verzeichnisstruktur. Die Zusammenstellung dieser Menge folgt lediglich administrativen Überlegungen und dient vor allem der Vereinfachung. Der Domänen Controller verwaltet dabei alle Aspekte der Anwender-Domänen-Interaktion, und verwendet die Informationen in der Sicherheitsdatenbank, um sich anmeldende Benutzer an Domänenkonten zu authentifizieren. Um dabei alle Konten einer Domäne zu verwalten, wird eine zentrale Verzeichnisdatenbank verwendet. Diese ist auf dem „Primary Domain Controller (PDC)“ abgelegt, die auf mehrere „Backup Domain Controller (BDC)“ repliziert werden kann [MS963].

Die Gruppierung einer oder mehrerer Domänen mit gemeinsamer Administration und Kommunikationsverbindungen untereinander ist für die Benutzer- und Ressourcenverwaltung in einem sogenannten „Domain Model“ zusammengefaßt [MS963].

Für das Internet bezieht sich der Begriff der Domäne auf der auf dem DNS basierenden Art der Adressierung, die mnemonische Bezeichner für Netzwerkkomponenten im Internet auf deren IP-Adresse abbildet. Gerade weil ein Computer, genauer seine Netzwerkadapter, im Internet mehrere kryptische IP-Adressen besitzen kann, ist diese anwenderfreundliche Namensgebung für die Lokalisierung von Netzwerkkomponenten innerhalb der Domäne, und damit auch im Internet, unumgänglich. DNS ist auf die Größe des Internets ausgelegt und erlaubt die Verwaltung von Host Bezeichnern, die sowohl im Internet als auch in privaten Intranet eindeutig sind.

Nicht zuletzt auch, weil die meisten leichten ORB auf JAVA Basis als Konsequenz aus denen noch zu erläuternden Sicherheitsrestriktionen der VM auf einen vollständigen Naming Service verzichten und statt dessen direkt auf DNS aufsetzen, soll kurz auf dessen Struktur eingegangen werden. Das konzeptionelle Benennungssystem des DNS basiert auf einer hierarchischen und logischen Baumstruktur, die als „Domain Name Space" bezeichnet wird. Die *Wurzel* (Root) des Domain Name Space wird durch das „Internet Network Information Center (InterNIC)" verwaltet. InterNIC (http://www.internic.com) hat die Aufgabe, administrative Verantwortung in Teilen des Domain Name Space an Organisationen und Unternehmen, die an das Internet angeschlossen sind, zu delegieren [MOA96].

Diese Subadministratoren verwenden DNS Server, um die Abbildung von Namen auf IP-Adressen für Computer und Netzwerkkomponenten innerhalb ihrer Domäne zu verwalten. Server, die auf den oberen Ebenen und an den privaten unteren Domänenknoten der DNS Hierarchie eingesetzt werden, werden als „Authoritative Name Server" bezeichnet. Der Aufbau dieser Hierarchie ist eine top-down Struktur, in der die obersten drei Hauptgebiete von InterNIC verwaltet und unterteilt werden [MOA96]:

- Organisatorische Domänen sind durch einen 3-Zeichen-Code bezeichnet, der die Hauptfunktion des Unternehmens darstellt (COM, EDU, GOV, INT, MIL, NET und ORG).
- Geographische Domänen werden mit einem 2-Zeichen-Code auf Basis des ISO 3166 Standards bezeichnet (siehe z.B. http://www.eexpert.com/_gp_dng.htm).
- „Reverse Domain" ist eine besondere Domäne mit Bezeichnung in-addr.arpa, die für die Abbildung von IP-Adressen auf Namen verwendet wird („reverse name lookup").

WinNT Domänen können direkt auf DNS Server aufgesetzt und durch diese verwaltet werden. Dadurch schafft man eine homogene Struktur für Bezeichner, sowohl auf Basis des Betriebssystems als auch auf der des Netzwerkes. Diese Bezeichner können unmittelbar durch den Namensdienst eines „lightweight" TCP/IP-ORB verwendet werden.

3.2.3.1.2 Microsoft Interface Definition Language (MIDL)

Das Gegenstück zum DCE-IDL ist die „Microsoft Interface Definition Language (MIDL)". In der Entwicklungsumgebung von MS RPC V2.0 wurde MIDL um die Attribute, Anweisungen und Direktiven der „Object Description Language (ODL)" erweitert, so daß man mit Hilfe des MIDL-Compilers auch Typenbibliotheken für OLE Anwendungen erstellen kann. Grundsätzlich unterscheiden sich die beiden Sprachen zunächst im Namen ihrer Stub-Compiler. MIDL bietet darüber hinaus eine (marginal) erweiterte Spezifikation, kann aber in einem DCE Kompatibilitätsmodus konfiguriert werden. Syntaktische Unterschiede in den Datentypen der einzelnen Sprachen werden im Anhang E aufgeführt. Abschließend ruft der DCE-IDL-Compiler den C-Compiler automatisch auf, was bei dem MIDL-Compiler nicht der Fall ist. Diesem Unterschied ist in Skripten für die Kompilation (z.B. MakeFile) Rechnung zu tragen.

Ein besonderes Augenmerk fällt bei IDL auf die Übergabe von Zeigern, da sich die einzelnen Prozesse in unterschiedlichen Adreßräumen befinden. MS RPC V2.0 unterstützt alle drei OSF DCE Zeigertypen (vgl. Kapitel 3.2.2.3). Im Microsoft Modul des MIDL können im Gegensatz zu OSF DCE-IDL auch Zeiger in importierten IDL Dateien deklariert werden. Die folgende Reihenfolge von Zuständen wird dabei im sogenannten „Microsoft Extensions Mode" angewandt [MSRPC97]:

1. Ein explizites Pointer-Attribut ist auf Verwendungsseite deklariert.
2. Das REF Attribut, falls ein nicht attributierter Zeiger ein „top-level" Zeigerparameter ist.
3. Falls ein pointer_default Attribut in der definierenden Schnittstelle spezifiziert ist.
4. Falls ein pointer_default Attribut in der Basisschnittstelle spezifiziert ist.
5. Falls das UNIQUE Attribut vorhanden ist.

Abschließend unterstützt MIDL das Microsoft-proprietäre Transportprotokoll „Named Pipes“, welches besonders bei der Kommunikation von Microsoft Serverdiensten untereinander, aber auch in der einfachen Interprozeßkommunikation Einsatz findet. Letzteres wird noch in den Beispielimplementierungen vorgestellt.

Als Beispiel soll eine Schnittstellenspezifikation für ein SpellChecker Objekt erläutert werden.

```
[object, uuid(E7CD0D00-1827-11CF-9946-44553540000)]

interface ISpellChecker : IUnknown {
  import “unknwn.idl“;
  HRESULT LookUpWord ([in] OLECHAR word[31], [out] boolean *found);
  HRESULT AddToDictionalry([in] OLECHAR word[31]);
  HRESULT RemoveFromDictionary ([in] OLECHAR word[31]);
}
```

Die Schnittstellenspezifikation beginnt mit dem Schlüsselwort *object*, welches anzeigt, daß die MIDL Erweiterungen zur DCE-IDL, also der „Microsoft Extensions Mode“ verwendet wird. Es folgt die Schnittstellenbezeichnung, ein weltweit eindeutiger „Universal Unique Identifier (UUID)“. Diese Bezeichnung wurde im MS DCE aus OSF DCE übernommen. Weiter wird der Schnittstellenname ISpellChecker aufgeführt, der vom Interface IUnknown abgeleitet wird. Schon in diesem einfachen Beispiel sieht man gut, wie jede Schnittstelle auf das IUnknown Interface aufbaut (siehe dazu auch Anhang A und Abb. 22). Abschließend fällt noch auf, daß jede Funktion den Fehlerrückgabewert HRESULT zurück geben muß.

Sobald die Implementierung einer so spezifizierten Schnittstelle veröffentlicht ist, dürfen keine Änderungen am Objekt (also am Binärcode) mehr vorgenommen werden. Falls eine neue Methode veröffentlicht werden soll, muß ein neue Schnittstelle geschrieben werden, welche von der existierenden abgeleitet werden kann. Es bleibt aber mit einer neuen Interfacebezeichnung eindeutig. Auf diesem Wege erhält MS DCE eine persistente Schnittstellendefinition. Da, wie noch zu zeigen ist, eine Vererbung von Objekten selbst nicht möglich ist, kann auf diesem Weg der Aggregation von Schnittstellen eine Abwärtskompatibilität von Objekten im COM erreicht werden.

3.2.3.1.3 MS Remote Procedure Call (MS RPC)

Da MS RPC in das umgebende Betriebssystem integriert ist, werden alle Dienste, die unter OSF DCE separat definiert sind, hier vom Betriebssystem erfragt. So ist zum Beispiel auch verteilte Sicherheit für MS RPC verfügbar, da unter WinNT sowohl die Netzwerkkommunikation als auch das Dateisystem, als Teile des Betriebssystems, mit entsprechenden Sicherungsmechanismen ausgestattet sind.

Ein grundlegender Unterschied zwischen OSF DCE und MS DCE liegt in der Namensgebung der RPC-API Funktionen. RPC Funktionsnamen bestehen aus dem Präfix „Rpc“, einem Objektnamen und einem Verb, das eine Operation auf dem Objekt beschreibt (siehe [MSRPC97] für Ausnahmen).

Rpc*ObjectOperation*

Objekt: Ein Bezeichner, der ein RPC Objekt spezifiziert; eine Datenstruktur die durch eine RPC Funktion definiert ist.

Operation: Bezeichnet eine Operation auf dem Objekt, welches durch *Object* bezeichnet wird.

Dabei werden die Bezeichner für Objekte entsprechend der folgenden Tabelle abgekürzt:

Objekt	*Objekt im Funktionsnamen*
Binding	handleBinding
Endpoint	Ep
Interface	If
Management	Mgmt
Name-service group entry	NsGroup
Name-service management	NsMgmt
Name-service profile entry	NsProfile
Name-service server entry	NsBinding
Network	Network
Object, type UUID mapping	Object
Protocol-sequence vector	ProtseqVector
Server	Server
String	String
String binding	StringBinding
UUID	Uuid

Tab. 10: Namenskonvention für MS RPC-API Funktionen.

Funktionen, die auf UUID-Objekten operieren, werden nicht mit dem Präfix „Rpc“ sondern mit „Uuid“ bezeichnet.

Während frühere Versionen des MS RPC auf die integrierte Sicherheit des „Named Pipe“ Transportprotokolls aufsetzten, implementiert die vorliegende Version eine transportunabhängige Sicherungsfunktion von OSF DCE RPC. Dabei wird der WinNT Sicherheitsdienst als Standard Sicherheitsanbieter verwendet. Dieser Schutz auf hoher Ebene ermöglicht es Servern, Klienten-anforderungen auf Basis von authentifizierten Identitäten, die mit jeder einzelnen Anforderung assoziiert sind, zu filtern. Dieser Mechanismus wird als *authentifizierter RPC* bezeichnet.

Um authentifizierte RPCs verwenden zu können, übergibt der Klient Sicherheitsinformationen seines Benutzers an die Laufzeitbibliothek. Diese Informationen werden als „*Credential*“ bezeichnet. Die Laufzeitbibliothek des Klienten übergibt diese Credentials an die Laufzeit-bibliothek des Servers, der diese dann an den Sicherheitsdienst übergibt. Dieser überprüft an-schließend die Gültigkeit. Sind die Credentials gültig, führt der Server den Aufruf aus, ansonsten wird der Klient abgelehnt.

Folgende Aufgaben werden vom Server ausgeführt, wenn der Klient eine Anbindung unternimmt:

1. Verknüpfungsinformationen über den Klienten werden erfragt.
2. Über die erhaltene Verknüpfung werden Authentifizierungsinformationen erfragt und mit Hilfe des Sicherungsdienstes überprüft.
3. Die Credentials des Klienten werden mir der ACL auf der Datenbank des Sicherheitsservers verglichen.

Mit der Version 2.0 des MS RPC werden jetzt auch IPX und UDP mit Sicherungsmechanismen unterstützt. Die Laufzeitbibliothek unterstützt weiter das Banyan Vines SSP Transportprotokoll[15] [SIN97]. DCE Pipes werden auf allen Transportprotokollen unterstützt. Abschließend soll noch darauf hingewiesen werden, daß sich in der aktuellen Version des MS RPC die Stubs nicht mehr auf die Existenz der Betriebssystem API-Methoden `MIDL_User_Allocate()` und `MIDL_User_Free()` stützen müssen, um Speicher auf Seite des Servers zu allokieren und freizugeben. Die Version 2.0 des MS RPC ist in diesem vormals wichtigstem Unterschied [ROS931] der Systeme konform zum OSF DCE geworden (vgl. Anhang E).

[15] Banyan Enterprise Klient für WinNT V5.56 erforderlich.

Ein wesentlicher Unterschied bleibt jedoch bestehen. ShutDown Nachrichten durch den DCE-Server werden bei MS DCE ignoriert. Eine Microsoft Erweiterung im MIDL, die `RpcMgmtEnableIdleCleanup()` Funktion, gibt unbenutzte Ressourcen wie Netzwerkverbindungen auf der Klientenseite frei. Verbindungsorientierte Protokolle setzen dabei ein fünf Minuten Timeout als vorgegebene Wartezeit ein, um festzustellen, ob eine Ressource noch benötigt wird. Dieses Verfahren verfolgt zwar dieselbe Grundidee, ist aber in der Implementierung dynamischer [ROS932].

Ein Großteil der Unterschiede in den RPC Implementierungen kann durch Header Dateien abgedeckt werden [ROS932].

3.2.3.1.4 WinNT Dienste

Da wir in diesem Umfeld WinNT als VS MS DCE betrachten wollen, können wir auch dessen Dienste in die bereits gewohnten Schemata einordnen. Obwohl die im folgenden aufgeführten Dienste zum Teil weniger zum VS selbst gehören, als mehr durch das Betriebssystem immanent angeboten werden, ermöglicht diese Art der Gegenüberstellung einen besseren Vergleich besonders zwischen OSF DCE und MS DCE.

WinNT unterstützt verschiedene Sicherheitsprotokolle, da jedes einzelne Protokoll entweder Kompatibilität mit bestehenden Klienten, Sicherheitsvorteile gegenüber seinen Konkurrenten oder Interoperabilität für heterogene Netzwerke wie dem Internet bietet. Damit beschränkt die WinNT Architektur nicht die Verwendung von Sicherheitsprotokollen. Unter Verwendung der Win32 Sicherheits-API werden die Anwendungen von den Unterschieden einzelner Sicherungsprotokolle isoliert. Anwendungsschnittstellen höherer Ebene, die durch authentifizierte RPC und COM angeboten werden, bieten einen hohen Abstraktionsgrad, so daß Sicherungsdienste auf Basis von Schnittstellenparameter verwendet werden können. Die WinNT Sicherheitsinfrastruktur unterstützt die drei wichtigsten Sicherheitsprotokolle [SIN97]:

- Das „WinNT LAN Manager (NTLM)“ Authentifizierungsprotokoll wird von WinNT 4.0 und früheren Versionen verwendet. NTLM wird auch weiterhin von Microsoft unterstützt werden und wird für eine „pass-through“ Netzwerkauthentifizierung, entfernten Dateizugriffen und für authentifizierte RPCs unter früheren Version von WinNT verwendet.
- Das Kerberos V5 Authentifizierungsprotokoll ersetzt NTLM als das vorrangige Sicherungsprotokoll für den Zugriff auf Ressourcen innerhalb und zwischen WinNT Domänen. Dieses Protokoll ist ein gereifter Industriestandard. Einige Vorteile sind wechselseitige Authentifizierung von sowohl Klienten als auch Server, reduzierte Serverlast während des Verbindungsaufbaus und die Unterstützung von Delegation und Autorisation von Klienten an Server unter Verwendung von Stellvertretermechanismen.
- Die „Distributed Password Authentication (DPA)“ ist das gemeinsame Authentifizierungsprotokoll, welches von den größten Internet Mitgliederorganisationen wie MSN und CompuServe verwendet wird. Es hat insbesondere den Vorteil, daß Anwender immer mit demselben Paßwort auf alle Server einer Mitgliederorganisation zugreifen können.

Protokolle auf Basis von öffentlichen Schlüsseln bieten Schutz der Privatsphäre und Zuverlässigkeit im Internet. SSL (siehe auch Anhang B) ist zur Zeit der de facto Standard für Verbindungen zwischen Internet-Browsern und -Servern im Internet (Eine IETF Protokolldefinition auf Basis von SSL3 ist unter der Bezeichnung „Transport Layer Security Protocol (TLS)“ angekündigt). WinNT 4.0 bietet Sicherungsmechanismen, die das SSL/PCT Protokoll als sicheren Kanal implementieren.

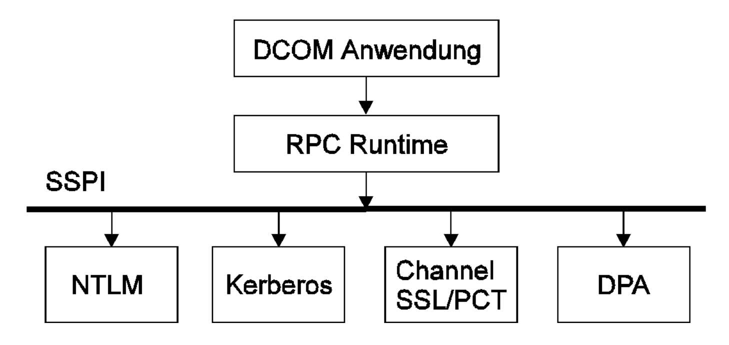


Abb. 18: Modell des WinNT „Security Support Provider Interface“.

Das Kerberos Authentifizierungsprotokoll definiert die Interaktion zwischen einem Klienten und einem Netzwerk Authentifizierungsdienst, der als „Key Distribution Centre (KDC)“ bezeichnet wird. WinNT implementiert einen KDC als Authentifizierungsdienst auf jedem Domänen Controller. Diese Domäne ist das Äquivalent zur einer Kerberos- oder OSF DCE-Zelle. Die WinNT Implementierung basiert auf der Definition des Kerberos Protokolls im Internet [RFC 1510]. Die Laufzeitumgebung des Kerberos-Klienten ist als WinNT Sicherheitsanbieter implementiert, der auf dem „Security Support Provider Interface (SSPI)“ basiert. Initiale Kerberos-Authentifizierung ist in die WinNT-Anmeldung integriert. Der Kerberos Server (KDC) ist im WinNT Sicherheitsdienst integriert, der auf dem Domänen Controller läuft und den WinNT Verzeichnisdienst als Datenbank für die Kontenverwaltung von Principals und -gruppen verwendet [MS966].

Das Kerberos Authentifizierungsprotokoll erweitert die unterliegenden Sicherheitsmechanismen von WinNT und stellt folgende Eigenschaften zur Verfügung [MS966]:

- Schnelle Authentifizierung während der Initialverbindung, da der Domänen Controller die Anwendungsserver entlastet.
- Delegation der Authentifizierung in vielschichtigen Klienten- / Serveranwendungen. Der Server verkörpert den Klienten nach dessen Authentifizierung. Dadurch kann sich der Server seinerseits bei Bedarf am nächsten Server authentifizieren, der wiederum den Klienten verkörpert.
- Transitive Vertrauensbeziehungen für domänenübergreifende Authentifizierung. Jeder Klient kann sich bei beliebigen Domänen in der Domänenstruktur authentifizieren, da der KDC Bescheinigungen anderer KDC innerhalb der Struktur akzeptiert. Dadurch wird die Administration von großen Netzwerken vereinfacht.

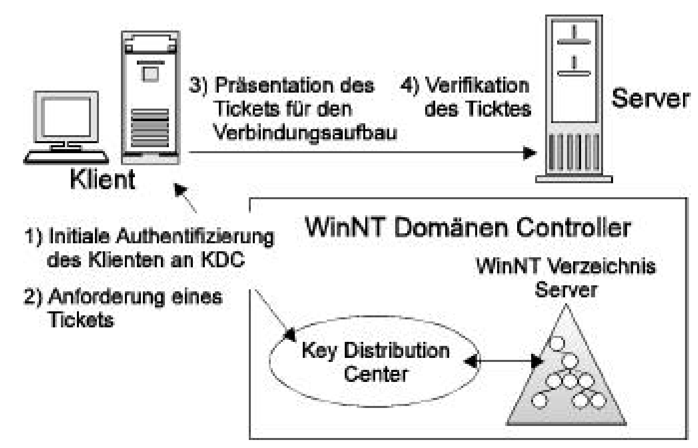


Abb. 19: Überblick über das Kerberos Authentifizierungsprotokoll unter WinNT.

MS DCE bietet einen eigenen, proprietären und sehr einfachen Namensdienst, den *Locator*. Im Unterschied zum DCE Verzeichnisdienst ist der Namensraum des Locator flach anstatt hierarchisch. Der WinNT 3.1 RPC-Locator unterstützt so keine Gruppen und damit auch nicht

die entsprechenden API (gekennzeichnet durch das Präfix „RpcNsGroup“, vgl. Anhang E). Er ermöglicht verteilten Anwendungen den RPC-Namensdienst zu benutzen, und verwaltet die Datenbank des RPC-Namensdienstes. Dabei führt er eine verzögert konsistente Datenbank, die strikt zwischengespeichert wird und flüchtig ist [MSKB971].

Die MS RPC-Laufzeitbibliothek bietet ein Menge von Namensdienst-APIs (gekennzeichnet durch das Präfix „RpcNs“) an, die Anwendungen auf RPC-Basis den Zugriff auf Bezeichnungsinformationen ermöglichen. RPC-Server können Namen und die damit assoziierte Verknüpfungsinformationen an einen Namensdienst weitergeben und abfragen. Die RPC-Bibliothek implementiert diese API indem sie mittels des RPC Runtime Interface „Name-Service Independent (NSI)“ auf die Datenbank des Namensdienstes zugreift. Diese Schnittstelle kann durch verschiedene Name Service Anbieter wie zum Beispiel RPC-Locator und DCE CDS verwendet werden [ROS932].

Wenn DCE CDS als RPC-Namensdienst konfiguriert wird, können zwar WinNT Server ihre Verknüpfungsinformationen an den CDS exportieren und sie damit den RPC Klienten zur Verfügung stellen; die umgekehrte Richtung aber, also der Import von Verknüpfungsdaten vom CDS in den RPC-Locator ist allerdings nicht vorgesehen. Die Entscheidung, welcher Namensdienst einzusetzen ist, wird auf Administrationsebene getroffen [ROS932]. Der Quellcode der Anwendung selbst bleibt von dieser Entscheidung unbeeinflußt, da die Unterschiede in den NSI Routinen implementiert sind, die den Klientenaufruf an den Server ausführen. Bei der Verwendung von verteilten Objekten auf Basis des COM empfiehlt sich aber die Auswahl des einfacheren Namensdienstes von WinNT, da auch abseits des DCE CDS Konfigurationen für die COM-Komponenten auf Seiten von WinNT vorgenommen werden müssen [DEC94].

Für die Verwendung des DCE CDS, anstelle des Locators, müssen folgende Einträge in die Registry, der Konfigurationsdatenbank (Registry) von WinNT, eingefügt werden:

```
HKEY_LOCAL_MACHINE/Software/MS/Rpc/Name Service/NetworkAddress
HKEY_LOCAL_MACHINE/Software/MS/Rpc/Name Service/Endpoint
```

Da eine MS RPC Anwendung keine vermittelnde Instanz wie den DCE Clerk zur Verfügung stellt, wird dessen Funktion in einem Stellvertreteragenten realisiert, der als „Name Service Interface Daemon (NSID)“ bezeichnet wird. Der NSDI läuft auf einem DCE System und verarbeitet NSI Aufrufe von MS RPC-Systemen. Dieser Zusammenhang wird in der folgenden Abbildung deutlich [ROS932].

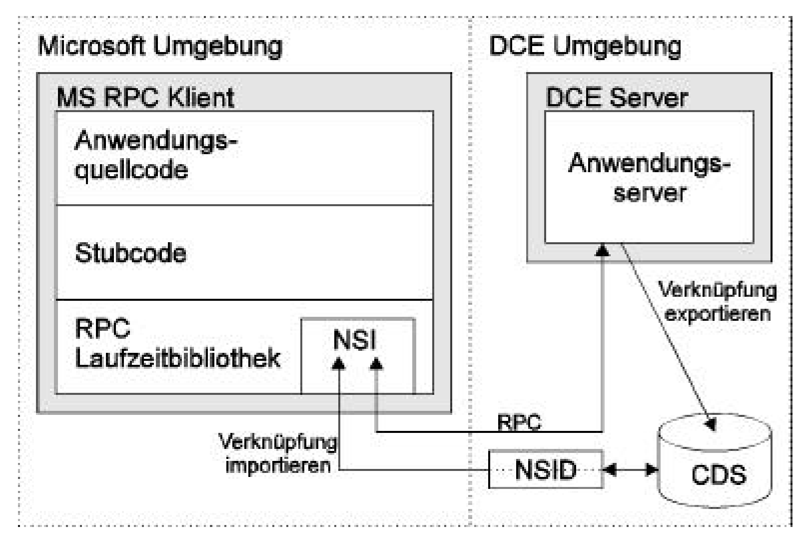


Abb. 20: Domänenübergreifender DCE CDS Namensdienst.

Erfahrungswerte zeigen, daß der NSDI hier leicht einen Flaschenhals darstellen kann. Je nach Verkehrsaufkommen können nur ca. 100 Klienten performant bedient werden. Insbesondere bei Zugriffen aus dem Internet kann es hier zu Überläufen kommen [ROS932].

Microsoft bietet ein einheitliches Dateisystem in WinNT, das auch MS-DOS abdeckt. Dieses Dateisystem ist nicht mit dem DCE DFS kompatibel. Damit wird die Möglichkeit der Verwaltung großer Namenräume stark eingeschränkt.

3.2.3.1.5 Threads

Threads stehen unter WinNT als Bestandteil des Betriebssystems zur Verfügung. Die WinNT Implementierung setzt nicht auf den POSIX 1003.4 Standard auf (vgl. Kapitel 3.2.2.6), was bei Portierung auf OSF DCE-Server eine Änderung am Server-Betriebssystem erforderlich macht. Weiter werden Threads nicht auf den Betriebssystemen MS-DOS, Win3x und Win95 angeboten, was den Einsatz von MS DCE-Servern auf diesen Systemen, aus Leistungsüberlegungen, praktisch ausschließt. Diese Überlegung deckt sich mit der allgemeinen Strategie des Unternehmens, die nicht WinNT Systeme als dedizierte Klientensysteme betrachtet. Die Portierung von Threads auf die Win32-API bietet allerdings Ausweichmöglichkeiten [ROS932].

Der ursprüngliche Ansatz, der die Erstellung von multithreaded Komponenten erlaubt, wird in MS DCE als *apartment model* bezeichnet. Hintergrund ist, daß obwohl ein Prozeß multithreaded implementiert sein kann, dies für einzelne Komponenten nicht möglich ist. Jeder Thread arbeitet innerhalb eines „Appartements“ und jede Komponente existiert in nur einem solchen abgegrenzten Adreßraum [SIN97].

In MS DCE V2[16] steht *free threading* zur Verfügung, was die thread-übergreifende Referenzierung von Komponenten erlaubt. Dadurch wir die Anzahl von aktiven Komponenten drastisch reduziert und das gesamte System entlastet [CHAP96].

3.2.3.2 COM als Middleware für verteilte Objekte

3.2.3.2.1 Entwicklung von OLE zu COM/DCOM

Die an „Objekten orientierte” Sichtweise von Interprogrammkommunikation hat bei der Firma Microsoft eine lange und wechselhafte Geschichte.

Die Wurzel der Entwicklung wurde 1991 unter dem Begriff “Object Linking and Embedding 1” (OLE) bekannt, und stellte eine Basistechnologie für Komponentendokumente dar. Der Gedanke, der dieser Entwicklung zu Grunde lag war der Versuch, dem Anwender eine dokumentenorientierte Sicht auf die EDV zu geben.

Nachdem man zu der Erkenntnis gekommen war, daß das Problem der Komponentendokumente ein Teilproblem der Kommunikation von Softwarekomponenten allgemein war, wurde 1993 „Object Linking and Embedding 2 (OLE 2)“ eingeführt. Diese, zunächst als „BugFix” für OLE 1 gedachte Erweiterung, basierte schon auf dem „Component Object Model (COM)”. Es war schnell klar, daß OLE 2 mehr als nur Dokumente von verschiedenen Programmen kombinieren konnte. COM stellt ein allgemeines Paradigma für die Kommunikation von Softwarekomponenten wie zum Beispiel Programmbibliotheken, Anwendungen und Systemsoftware dar. Losgelöst von der dokumentenorientierten Sichtweise, wurde schnell der neue Terminus OLE (der keine Abkürzung ist) benannt, der (fast alle) COM-basierten Technologien zusammenfaßte [CHAP96].

Die zur Zeit letzte Etappe dieser Entwicklung bildet seit Anfang 1996 der Begriff ActiveX. Zunächst wurde der neue Terminus mit Technologien von Microsoft assoziiert, die mit dem Internet in Verbindung oder in diesem angewandt werden konnten. Diese Auffassung ist noch heute weit verbreitet und führt damit oft zu Mißverständnissen bei der Diskussion um den Begriff und dessen Technologie. Da auch ActiveX auf COM basiert, stand der Terminus in direkter Ver-

[16] Unter WinNT 4.0.

bindung zu OLE. Die dadurch entstandene Unklarheit seitens der Anwender begegnete Microsoft schnell mit der Neudefinition der Begriffe: Zur Zeit bedeutet OLE wieder eine Technologie zur Erzeugung von Komponentendokumenten unter Verwendung von „Object Linking and Embedding". Die Sammlung von COM-basierten Technologien der Firma wurde unter dem Sammelbegriff ActiveX zusammengefaßt.

Seit 1996 steht auch die Erweiterung DCOM zur Verfügung, die eine Implementierung von COM-Komponenten auf VS ermöglicht. Im folgenden wird COM mit DCOM synonym verwendet, da eine inhaltliche Trennung nicht sinnvoll und nur historisch bedingt ist. Die noch zu erläuternde Ortstransparenz von Komponentenobjekten beruht lediglich auf der Mächtigkeit der verwendeten COM-Bibliothek, nicht aber auf dem Modell selbst. Die Erweiterungen der COM-Bibliothek, die das DCOM ausmachen, werden in dem separaten Kapitel 3.2.3.2.3 behandelt. Der Begriff DCOM wird nur dann verwendet werden, wenn ausdrücklich die Erweiterung zu COM gemeint ist.

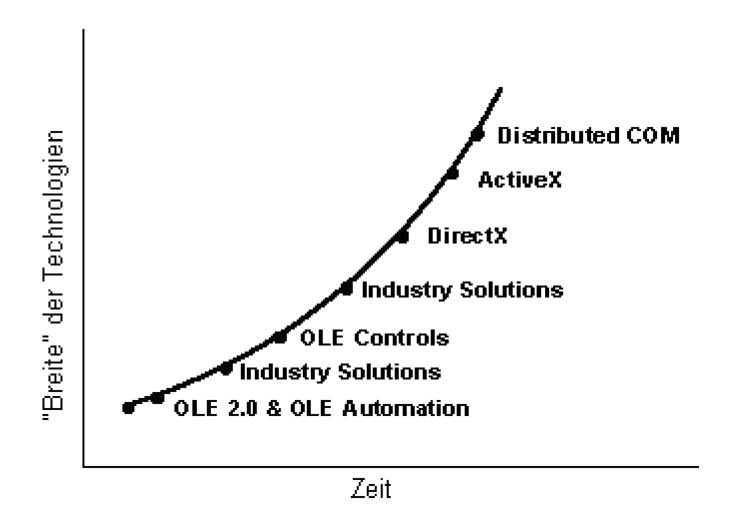


Abb. 21: Entwicklung von OLE zu COM/DCOM [STE96].

Diese Abbildung[17] verdeutlicht die technologische Entwicklung der Komponentenmodelle der Firma Microsoft. Als letzter Glied dieser Kette ist für das kommende Jahr COM+ als „Rahmenwerk angekündigt, mit dessen Hilfe die Grenzen zwischen lokalem und entfernten Rechnern verwischt werden sollen [BEI97]" (vgl. Kapitel 5.4).

Ein Ende dieser Entwicklung ist nicht abzusehen. Allein die feste Basis der oben genannten Technologien ist COM. Dadurch wird eine nähere Betrachtung in dieser Arbeit sinnvoll.

3.2.3.2.2 Das Component Object Model (COM)

3.2.3.2.2.1 Allgemeines

Um die grundsätzliche Anforderung der Kommunikation zwischen (Teil-) Anwendungen zu vereinfachen, wurde von Microsoft 1993 das „Component Object Model" als Plattform für OLE 2 eingeführt. COM wurde zu diesem Zeitpunkt wie folgt definiert:

> ***COM** definiert eine einheitliche Schnittstelle, um Software Dienste anzusprechen.*

Im COM ist ein Objekt kompilierter Code, der dem Rest des Systems einen Dienst zur Verfügung stellt. Um Verwechslungen mit der Namensgebung in der objektorientierten Programmierung zu

[17] DirectX ist eine Schnittstelle zwischen Hard- und Software, die Programmen, auf Basis von MS Windows, Echtzeitzugriffe auf die unterliegende Hardware ermöglicht.

vermeiden, wird ein COM-Objekt als „Komponentenobjekt“ oder als *Komponente* bezeichnet [KIN95].

Jedes COM-Objekt unterstützt mindestens ein *Interface* (~ Schnittstelle, im weiteren wird aber der von Microsoft eingeführte Terminus verwandt, um eine Differenzierung zu anderen Middleware-Technologien wie CORBA zu ermöglichen), das wiederum mindestens eine Methode realisiert.

> *Ein **Interface** ist die Definition des erwarteten Verhaltens und der erwarteten Zuständigkeiten eines Komponentenobjekts.*

Eine *Methode* besteht aus einer Funktion oder Prozedur, die eine bestimmte Aktion ausführt. Methoden innerhalb eines Interfaces sind in der Regel inhaltlich verwandt. Wichtig ist, daß ein Interface weder eine Klasse, noch ein Komponentenobjekt ist, da es weder instantiiert werden kann noch eine Implementierung besitzt.

Im Gegensatz zu zum Beispiel C++ erfolgt der Zugriff auf die Dienste einer COM-Komponente ausschließlich über die Interfaces - die Objektdaten können nie direkt adressiert werden. Dies ist ein fundamentaler Baustein der COM-Architektur. Dadurch wird eine vollständige Kapselung der Daten und deren Verarbeitung erreicht. Dies ist eine Basisanforderung für einen „echten” Standard für Komponentensoftware. Dadurch wird auch Ortstransparenz erreicht (prozeß- oder netzwerkübergreifende Aufrufe), da der gesamte Datenzugriff über Interfaces realisiert ist, die auch in einem lokalen Stellvertreterobjekt zur Verfügung gestellt werden könnten. Dieses leitet die Aufrufe lediglich weiter und schickt die Antworten zurück.

Ist COM objektorientiert? Im Umfeld der OOP besteht ein Objekt aus zwei Elementen: eine definierte Menge von Daten (Zustand oder Attributen) und einer Gruppe von Methoden. Dies trifft auf Komponentenobjekte zu, obwohl diese im allgemeinen mehrere Interfaces unterstützen. Wie auch im allgemeinen Verständnis ist in COM eine Klasse wie folgt zu definieren:

> *Eine COM **Klasse** identifiziert eine bestimmte Implementierung einer Menge von Interfaces.*

Um „objektorientiert” zu sein, sollte ein Modell folgende Eigenschaften unterstützen: Kapselung, Polymorphismus und Vererbung [HER93]. Die Daten eines Komponentenobjekts sind durch die Interfaces verkapselt. Weiter wird Polymorphismus unterstützt, da zwei verschiedene Komponentenobjekte das gleiche Interface (-bezeichnung) oder nur eine gleiche Methode anbieten können. Der einzige Unterschied ist, daß COM lediglich die Interfacevererbung, nicht aber die Komponentenvererbung unterstützt. Diese Einschränkung ist ein Tribut an die strenge Kapselung der Objektimplementierung im COM, da durch Vererbung der Implementierung dem Erben (unter Umständen geheime) Eigenschaften des Ursprungsobjekts offenbart werden können. Die Unterstützung der Interfacevererbung birgt diese Gefahr nicht. Um trotzdem den Code von Komponenten wiederverwenden zu können, simuliert COM Codevererbung durch Kapselung und Aggregation. Durch *Kapselung* kann ein Komponentenobjekt die Methoden eines zweiten Komponentenobjektes aufrufen, um eine Aufgabe zu erfüllen. Durch *Aggregation* kann ein Komponentenobjekt Methoden eines zweiten Komponentenobjekts als seine eigenen offenlegen. Im Vergleich zu den traditionellen Programmiersprachen, wie zum Beispiel C++, ist COM nicht objektorientiert, adressiert aber auch andere Probleme als die der Programmiersprachen. Als Modell bietet COM alle Vorteile der Objektorientierung.

Ein COM-Objekt wird immer innerhalb eines Servers implementiert, der entweder als eigenständiger Prozeß oder als „Dynamic-Link Library (DLL)“ realisiert werden kann. Eine DLL ist eine ausführbare Routine, die gewöhnlich eine bestimmte Funktion oder eine Reihe von Funktionen ausführt. Sie wird in separaten Dateien (mit der Dateinamenserweiterung DLL) gespeichert und kann bei Bedarf von dem Programm geladen werden, das sie aufruft. Einmal geladen, verbleibt die DLL im Arbeitsspeicher.

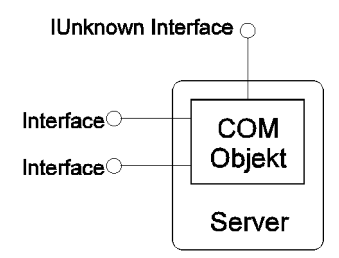


Abb. 22: Interfaces eines Komponentenobjekts innerhalb eines Servers [CHAP96].

Jede COM-Komponente ist die Instanz einer Klasse. Um eine Methode einer COM-Komponente aufzurufen, muß der Klient einen Zeiger auf dieses Interface generieren. Dafür kann der COM-Bibliothek eine Anfrage mit dem Klassennamen und der Interfacebezeichnung übergeben werden. Daraufhin initiiert die COM-Bibliothek bei Bedarf den entsprechenden Server der COM-Komponente und gibt einen Zeiger auf das angeforderte Interface zurück. Die COM-Bibliothek ist die Basis einer jeden COM Implementierung und Teil des Betriebssystems. Die Dienste der COM-Bibliothek werden nicht als COM-Interface sondern als API angeboten [CHAP96].

Aufgrund dieser Anforderung läßt sich abschließend auch eine Definition für COM-Komponenten formulieren (die Bezeichnung ActiveX-Komponente wird in der Literatur oft synonym verwandt, ist aber aufgrund der Definition nicht sinnvoll):

> *Ein **Komponentenobjekt** ist eine Softwarekomponente, die das Interface IUnknown unterstützt und die zur Laufzeit die Instantiierung einer Klasse bildet [ORF97].*

Die Bedeutung des IUnknown Interfaces wird im folgenden Abschnitt verdeutlicht (vgl. auch Anhang A).

3.2.3.2.2.2 COM-Interfaces

Um das Interface eines Komponentenobjekts identifizieren zu können, hat jedes Interface zwei Bezeichner. Der erste, nicht zwingend eindeutige Bezeichner, bildet einen mnemonischen Namen auf ein Interface ab. Die Notwendigkeit für eine weltweit eindeutige, dafür aber sehr unleserliche Bezeichnung wird durch einen „globally unique identifier (GUID)“ erfüllt, der aus dem OSF DCE übernommen wurde. Diese wird durch den Entwickler einem Interface zugewiesen, und besteht aus einem 16-byte Wert der aus „Medium Access Control (MAC)“-Adresse [HAL96] und einem Zeitstempel generiert wird. GUID für Interfaces werden als „interface identifier (IID)“ bezeichnet. Die Gefahr einer Entschlüsselung des IID, um auf die Netzadresse des Objektservers schließen zu können, besteht nicht; das System, auf dem ein Komponentenobjekt aktiv ist, stimmt nur in den seltensten Fällen mit dem System seiner Erstellung überein. Damit ist die „erbeutete“ Information ohne jede Bedeutung [CHAP96].

Per Konvention beginnen Interfacebezeichnungen mit einem “I”.

Das wichtigste Interface ist *IUnknown,* das von jedem Komponentenobjekt zur Verfügung gestellt wird. Durch diese Schnittstelle kann die Lebenszeit der Komponente überwacht und die Namen der Interfaces mit deren Zeiger erfragt werden (s. Anhang A).

Im Gegensatz zu CORBA unterstützt ein Komponentenobjekt selbst keine Identifikation (im CORBA-Terminus „Object Reference”). Nur die Interfaces selbst werden durch die IID referenziert. Die Interface Implementierung unterliegt in COM einem vorgegeben Standard. Durch standardisierte Binärformate von Komponenten können Klienten die Methoden einer Komponente unabhängig von der verwendetet Programmiersprache verwenden.

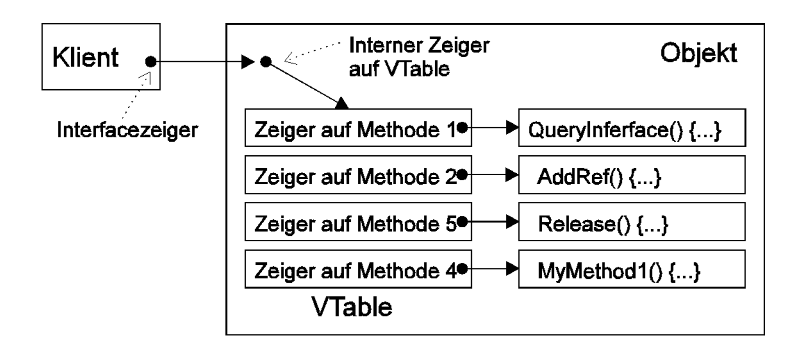


Abb. 23: VTable in COM-Komponenten [CHAP96].

Die Realisierung erfolgt, indem der Interfacezeiger des Klienten auf einen Zeiger innerhalb der Komponente verweist. Dieser zeigt wiederum auf die *VTable*, einer Liste von Zeigern auf die Methoden der Komponente (vgl. Abb. 23). Die ersten drei Methoden (QueryInterface, AddRef und Release) sind durch IUnknown definiert und, durch die Vererbung, Teil der Komponente. Diese Binärstruktur existiert für jedes Interface, welches von einer Komponente unterstützt wird [CHAP96].

Solche Aufrufe sind natürlich für Entwicklungsumgebungen wie MS Visual Basic nicht geeignet, da keine Zeiger unterstützt werden. Hier wird auf das *IDispatch* Interface zurückgegriffen, welches die Komplexität der Zeigeroperationen verbirgt. Dieses IDispatch Interface ist ein normales COM-Interface. Wie jedes andere auch, verwendet es eine VTable, welche Zeiger auf die Methoden enthält. Der Unterschied zu normalen Interfaces besteht in der Methode `IDispatch::Invoke`. Mittels dieser Methode kann ein Klient direkt eine Gruppe von Methoden aufrufen und die erforderlichen Parameter übergeben. Dafür muß der Entwickler einer Komponente mit IDispatch-Unterstützung einzeln festlegen, welche Methode für diese direkten Aufrufe zur Verfügung stehen. Dies wird durch die Definition eines Dispatch Interfaces realisiert, welches allgemein auch als *Dispinterface* bezeichnet wird. Ein Dispinterface ersetzt also die VTable und macht die damit verbundene Zeigerarithmetik für den Entwickler transparent. Der Aufruf einer Methode erfolgt lediglich durch Aufruf von `Invoke()` und Übergabe des „Dispatch Identifier (DISPID)“, welcher jede Methode identifiziert, die durch das Dispinterface aufgerufen werden kann. Dieser Mechanismus erlaubt es auch Sprachen, wie MS Visual Basic, über die COM-Bibliothek Methoden aufzurufen, die damit nur Code für die Navigation durch das standardisierte Dispinterface zur Verfügung stellen müssen. Diese Aufgabe wird üblicherweise von der Laufzeitumgebung übernommen. Das folgende Beispiel verdeutlicht diese Transparenz für den Entwickler. Es ist die Realisierung einer Rechtschreibüberprüfung: Die Funktion `SpellCheck()` erhält einen Suchbegriff als Übergabewert und gibt TRUE zurück, wenn das Wort gefunden wurde, FALSE sonst.

```
Function SpellCheck(strTestWord As String) As Boolean
  DIM Obj As Object
  Set OBJ = CreateObject(„Word.Application“)
  SpellCheck = Obj.CheckSpelling(strTestWord)
End Function
```

Die COM-Komponente WORD mit der Methode `CheckSpelling()` kann ohne Angabe einer Identifizierung der Klasse und durch Verwendung des WinNT Locator, unabhängig von der Lokalität der Komponente, instantiiert werden. Nach dem transparenten Aufruf des Dispinterfaces, kann die öffentliche Methode der Komponente sofort aufgerufen werden. Wie in der Abb. 23 demonstriert, besteht das IUnknown Interface nur aus drei Grundoperationen:

QueryInterface — Nachdem ein Klient nach der Instantiierung des Objekts den ersten Interfacezeiger erhalten hat, können, durch den Aufruf von `IUnknown::QueryInteface`, die Zeiger auf die anderen Interfaces der Komponente angefordert werden. Die IID des angeforderten Interfaces wird von

	dem Klienten als Parameter übergeben. Wenn die Methode unterstützt wird, wird der entsprechende Interfacezeiger, sonst NIL, zurückgegeben.
AddRef	Diese Methode implementiert den Mechanismus, der in COM als *reference Counting* bezeichnet wird. Da kein Klient, der Referenzen auf eine Komponente hält nicht wissen kann, ob noch andere Klienten diese Komponente weiterhin verwenden, wird durch das Zählen von aktiven Referenzen auf der Serverseite die Laufzeit der Komponente überwacht. Dies ist ein bedeutender Unterschied zum CORBA-Modell, in dem der ORB die Lebenszeit der angeschlossenen Objekte überwacht.
Release	Nachdem ein Klient die Serverkomponente nicht mehr benötigt, ruft er `Release()` auf, um den Referenzzähler zu dekrementieren.

Häufig wird das QueryInterface als wichtigstes Interface in COM betitelt. Dies liegt daran, daß dieser sehr einfache Mechanismus ein sehr wichtiges und komplexes Problem löst: die Versionskontrolle. Exemplarisch verwendet eine bestehende Anwendung eine COM-Komponente. Wird diese Komponente um eine Funktion erweitert, kann die Anwendung wie gewohnt die VTable durch die `QueryInterface` Methode abfragen. Da das Interface einer Komponente nicht verändert werden darf, müssen keine Modifikationen an der Anwendung vorgenommen werden - sie kennt die neuen Methoden der Komponente nicht. Die neue Funktionalität wird in einem neuen Interface spezifiziert, das wiederum von einer neuen Version der Anwendung parallel verwendet werden kann. Das `QueryInterface` und die Unveränderlichkeit von COM-Interfaces erlauben, daß zwei Komponenten separat entwickelt und aktualisiert werden können, die Interaktion aber gewährleistet bleibt [CHAP96].

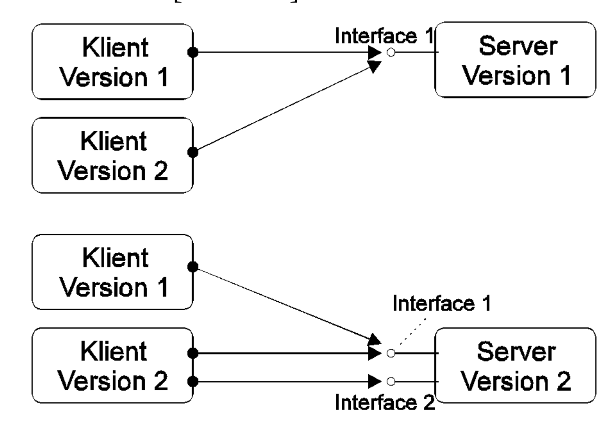


Abb. 24: COM Versionskontrolle durch Interfaces.

Die obige Grafik veranschaulicht diesen Sachverhalt.

Abschließend muß noch gesondert auf den Begriff der *Klasse* im Rahmen des COM eingegangen werden. Jede Komponente ist die Instanz einer Klasse und jeder Klasse kann eine eigene GUID zugewiesen werden, die als „Class Identifier (CLSID)“ bezeichnet wird. Ein CLSID ist somit ein universell eindeutiger Bezeichner (GUID), der den Typ eines Objektes identifiziert. Allerdings hat nicht jede Klasse zwingenderweise eine CLISID, da Komponenten einiger Klassen nicht durch die COM-Bibliothek erstellt werden und damit keine CLISD benötigen. Im COM bezeichnen CLSID eine bestimmte Implementierung einer Gruppe von Interfaces, nicht die Gruppe selbst. So ist es zum Beispiel unter COM nicht untersagt, daß einer Komponente Interfaces hinzugefügt werden, ohne die CLSID zu ändern. Die CLSID wird demnach lediglich dazu verwendet, einen bestimmten Code zu identifizieren, der durch die COM-Bibliothek geladen und für Objekte dieser Klasse ausgeführt wird. Jeder „normale“ Objekttyp hat seinen eigenen CLSID in der Registry Datenbank von MS Windows, so daß das entsprechende Objekt durch andere Programme geladen und programmiert werden kann.

3.2.3.2.2.3 COM Ortstransparenz

Dadurch, daß Objekte nur durch Zeiger auf Interfaces adressiert werden, verbirgt sich die eigentliche Lokation der Zielkomponente vor dem Klienten. Die Grafik zeigt die drei verwendeten Kommunikationsmechanismen.

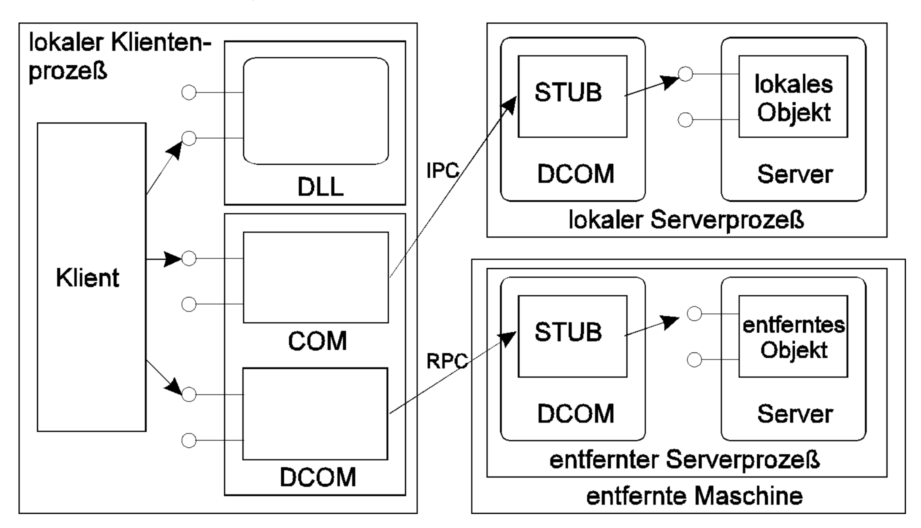


Abb. 25: Ortstransparenz im DCOM [CHAP96].

Damit gibt es drei mögliche Realisierungen für COM-Komponenten:

- „In-Process Server" werden als DLL realisiert und zur Laufzeit in den Speicherbereich des Prozesses geladen. Beachtenswert ist, daß eine DLL nur einmal pro Sitzung des Betriebssystems geladen wird. Obwohl sie in den exklusiven Speicherbereich des Prozesses geladen wird, kann das Betriebssystem bei Bedarf Kopien für andere Prozesse zur Verfügung stellen.
- Lokale Server werden über Interprozeßkommunikation („Inter-process Communication (IPC)") angesprochen, die durch einen sogenannten „Lightweight RPC" realisiert wird.
- Entfernte Server werden über den MS RPC angesprochen.

MS RPC ist ein Paket von API Funktionen, Bibliotheken und Programmen, die es Entwicklern ermöglichen, Klienten- / Serveranwendungen unter Vermeidung von Low-Level-Code zu generieren. MS RPC bietet dabei eine nahtlose Unterstützung der Netzwerkprotokolle Named Pipes, NetBIOS, TCP/IP, IPX/SPX und Lightweight RPC [HAL96]. Das Marshalling zur Überbrückung systemabhängiger Datenformate erfolgt durch Konvertierung aller übertragenen Daten in eine proprietäre „Network Data Representation (NDR)" [SIN97].

Abschließend soll kurz die Erstellung einer Objektinstanz unter COM demonstriert werden. Für eine weiterführende Betrachtung vergleiche [CHAP96]. Dieses „Bootstrap" Problem, also die Frage wie ein Klient den Zeiger auf eine angeforderte Komponente initial erhält, wird von der COM-Bibliothek gelöst. Diese Bibliothek muß von jedem System, das COM unterstützt, implementiert werden. Wichtigster Bestandteil dieser API ist die Möglichkeit eines Klienten einen Server zu starten. Die Gruppe von Funktionsaufrufen trägt das Präfix „Co", zum Beispiel `CoCreateInstance()`.

Fordert ein Klient die Instantiierung einer Komponente an, so übergibt er die entsprechende CLSID an die COM-Bibliothek. Anhand dieser eindeutigen Bezeichnung wird nun die sogenannte *Registry* verwendet, um die CLSID auf den tatsächlichen binären Servercode abzubilden. Die Klassen aller Objekte, die durch die COM-Bibliothek instantiierbar sein sollen, müssen in dieser (einfachen) Datenbank vorhanden sein. Das exakte Format der Registry wird durch COM nicht vorgegeben. Win95 und WinNT verwenden eine einfache Systemtabelle.

Diese Zuweisungstabelle muß auf allen Systemen mindestens folgende Elemente beinhalten [CHAP96]:

- Die CLSID, die als Schlüsselwert fungiert
- Eine Indikation des Servertyps (In-Process, lokal, entfernt)
- Für In-Process und lokale Server wird der vollständige Pfadname der Datei (DLL oder ausführbare Datei), für entfernte Server ein Suchkriterium (z.B. IP-Adresse oder NetBEUI Bezeichner) für das Auffinden des Hosts aufgeführt

Normalerweise fügt eine Klasse ihre eigene CLSID automatisch an die Registry an, sobald sie zum ersten Mal instantiiert wird. COM-Server besitzen per Konvention die Fähigkeit zur eigenen Registrierung. Bei Bedarf muß der Eintrag auch durch ein Installationsprogramm vorgenommen werden. ActiveX-Komponenten haben darüber hinaus immer die Möglichkeit der Eigen-registrierung.

3.2.3.2.2.4 Moniker

In objektorientierten Systemen ist es sinnvoll, eine Objektinstanz eindeutig identifizieren zu können. COM selbst bietet keine Möglichkeit eine Objektinstanz zu bezeichnen. Ein Klient kann eine generische Instanz einer Komponente durch den Aufruf von `CoCreateInsance()` und Übergabe der CLIDS erzeugen. Wenn dieses Objekt persistente Daten trägt, kann der Klient durch Verwendung der *IPersist* Interfaces das Objekt dazu veranlassen, diese Daten zu laden (für eine detaillierte Beschreibung des „Persistent Services“ von COM mittels des IPersist Interfaces siehe [CHAP96]). Dafür muß dem Klienten aber sowohl die CLSID als auch das Verfahren zum Laden der persistenten Daten bekannt sein. Die Kombination dieser Informationen wird deshalb in einer separaten Komponente gespeichert.

Der Begriff „Moniker” stammt aus dem englischen Sprachraum und bedeutet übersetzt Spitz-name. Diese Namensgebung umschreibt treffend die Definition:

> *Ein **Moniker** ist eine eindeutige Bezeichnung für eine bestimmte Objektinstanz, also eine eindeutige Kombination aus CLSID und persistenter Daten [CHAP96]. Die Abbildung von Moniker auf Objektinstanzen ist eindeutig.*

Aus der Sicht des COM ist ein Moniker selbst eine Komponente, welche das Interface IMoniker unterstützt. Darüber hinaus enthält jeder Moniker seine eigene persistente Datenmenge, die aus-reicht, um die von ihm benannte Objektinstanz zu starten und zu initialisieren.

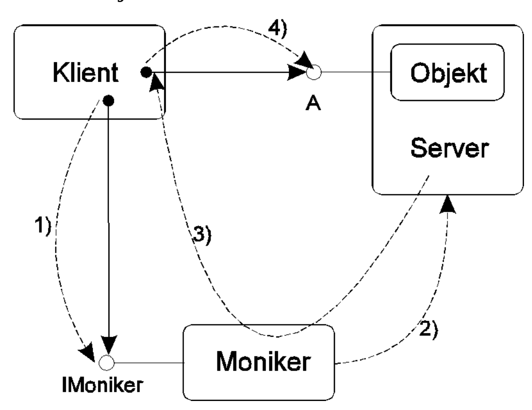


Abb. 26: Verwendung eines Monikers [CHAP96].

Die Verwendung eines Monikers wird in der obigen Abbildung dargestellt. Zunächst wird die `BindToObject()` Methode der IMoniker Schnittstelle durch den Klienten ausgeführt und als Parameter eine IID übergeben, die auf das Zielobjekt des Klienten verweist (s. 1). Anschließend instantiiert der Moniker das angeforderte Objekt und fordert es auf, die persistenten Daten zu

laden (s. 2). Der Zeiger auf das angeforderte Interface wird nun direkt von der Serverkomponente an den Klienten zurückgegeben (s. 3). Danach kann der Aufruf der Methode erfolgen (s. 4).

Damit nicht jeder Klient eine neue Instanz einer Komponente instantiiert, stellt MS DCE die „Running Object Table (ROT)" zur Geschwindigkeitssteigerung zur Verfügung. Hier können sich Komponenten registrieren, und damit als aktiv für die Umgebung kennzeichnen. Ein direkter Aufruf der aktiven Komponente über die Methode `IRunningObjectTable::GetObjekt` ist nun möglich, ohne daß eine neue Komponente langwierig von der COM-Bibliothek generiert werden muß. Diese Komponenten werden als *MultiUse Connector-Object* bezeichnet. Als Alternative können Server ihr Stammobjekt als *PublicNotCreatable* kennzeichnen, so daß Klienteninstanzen nicht direkt erstellen werden können [CHAP96].

3.2.3.2.3 Distributed Component Object Model (DCOM)

Von Anfang an war COM für verteilte Systeme ausgelegt. Dieser Abschnitt in der Architektur wurde mit dem „Distributed COM (DCOM)" 1996 technisch realisiert. DCOM fügt lediglich drei neue Elemente zur Basis von COM hinzu: Techniken für die Instantiierung entfernter Komponenten, ein Protokoll für deren Methodenaufruf und ein Mechanismus zur Sicherung des Zugriffs auf diese Komponenten. Damit ist eine Trennung im Sprachgebrauch unnötig.

Die Instantiierung einer Komponente erfolgt in COM durch Aufrufe der `CoCreateInstance()` Methode und anschließendem Abfragen der IID durch die `QueryInterface()` Methode. Dieser Mechanismus funktioniert auch für entfernte Komponenten im DCOM, allerdings muß zusätzlich auch die Maschine spezifiziert werden, auf der die Komponente ausgeführt werden soll. Diese kann durch Nachschlagen in der Registry anhand der CLISD gefunden werden. Wenn die entfernte Komponente als DLL realisiert ist, wird vom Betriebssystem (bei WinNT also von der Middleware) ein Ersatzprozeß gestartet, der die DLL lädt und ausführt. DCOM-Server werden einfach ausgeführt. Als Bezeichner für Maschine sind unter DCOM der DNS unter TCP/IP direkte IP-Adressen, NetBIOS-Namen und NetWare IPX/SPX-Bezeichner zugelassen.

Um die wiederholten Abfragen des QueryInterface zu vermeiden, wird unter DCOM die `CoCreateInstaceEx()` Methode angeboten. Im Gegensatz zu dem COM-Äquivalent können bei dieser Funktion eine Liste von IID mit einer Parameterübergabe übergeben werden. Alle angeforderten Interfacezeiger werden wiederum als Liste zurückgegeben. Darüber hinaus kann über `CoCreateInstanceEx()` auch der Ort der Ausführung spezifiziert werden. Anstatt auf die lokale Registry zu bauen, um das entfernte System zu lokalisieren, kann eine Maschine für die Ausführung der Komponente vom Klienten dynamisch bestimmt werden. Beide Mechanismen tragen erheblich zur Leistungssteigerung durch Lastverteilung bei verteilten Systemen bei. Die Verwendung von Moniker zur Instantiierung und persistenten Bezeichnung von Komponenten unter DCOM ist ebenfalls möglich [CHAP96].

Nachdem eine Komponente durch DCOM auf entfernten Systemen erstellt worden ist, liegt der zweite Teil der Aufgabe in der, für das System transparenten Kommunikation mit der erstellten Komponente. Sowohl MS RPC als auch dessen Verwendung in DCOM, die als „Object RPC (ORPC)" bezeichnet wird, unterstützen verbindungsorientierte Transportprotokolle, zum Beispiel für TCP-Verbindungen und verbindungslose Transportprotokolle, zum Beispiel für UDP. Unabhängig davon, welches Transportprotokoll im ORPC zum Einsatz kommt, muß der Klient Verbindungsinformationen (~ binding information) für die Zielkomponente besitzen, um einen ORPC Aufruf durchführen zu können. „Bootstrap" Verbindungsinformationen können durch `CoCreateInstanceEx()` oder durch Weitergabe zwischen Komponenten erhalten werden [CHAP96].

Ein Server, der eine oder mehrere Komponenten implementiert, „exportiert" Komponenten auf andere Systeme. Als Single-Threaded Prozeß agiert der gesamte Server als „Exporteur". Bei Multi-Threaded Servern, deren Komponenten durch Appartements getrennt sind, „exportiert"

jedes Appartement eine eigene Komponente. Jedem dieser Komponenten-Exporteure wird ein 8-Byte Wert zugewiesen, der als „Object Exporter Identifier (OXID)“ bezeichnet wird. Um eine entfernte Komponente ansprechen zu können, muß ein Klient also zunächst die OXID der Komponente erfragen. Auf jeder DCOM Maschine läuft ein sogenannter *OXID Resolver*, der die OXID auf die Verbindungsinformation der Komponente abbildet. Dieser veröffentlicht das Interface IObjectExporter[18], welches die Methoden `ResolveOxid()`, `SimplePing()` und `ComplexPing()` anbietet. Für eine ausführliche Darstellung dieser Methoden, die nicht zuletzt auch für die Überwachung der Lebenszeit von entfernten Komponenten verwendet werden, siehe [CHAP96].

Die Antwort auf die Frage, warum hier anstelle von OXID keine Moniker zum Einsatz kommen, liegt im NDR. Die NDR bietet eine standardisierte Abbildungsvorschrift um MIDL Datentypen zwischen Maschinen mit unterschiedlicher Datendarstellung zu übertragen. Interfacezeiger, die häufig in ORPC Aufrufen übergeben werden, sind aber nicht durch die NDR abgedeckt. Wenn ein Interfacezeiger auf eine entfernte Komponente verweist, muß eine sogenannte „Object Reference (OBJREF)“ übergeben werden. DCOM definiert die OBJREF mit folgenden Elementen:

- OXID
- Ein eindeutiger Interfacebezeichner für entfernte Interfaces, „Interface Pointer Identifier (IPID)“
- Ein eindeutiger Komponentenbezeichner, „Object Identifier (OID)“
- Verbindungsinformationen als Zeichenkette für den OXID Resolver der Maschine der Serverkomponente

Dieses Konstrukt kann durch die NDR übertragen werden, und enthält ausreichend Informationen, um durch den OXID Resolver die Verbindungsinformation des entfernten Interfacezeigers aufzulösen. Dadurch kann auch ein Resolver einen zweiten nach Verbindungsinformationen anfragen.

Abschließend erfolgt das Instantiieren einer lokalen Komponente bei COM durch den „Service Control Manager (SCM)“. Bei entfernten Komponenten gibt der SCM der lokalen Maschine eine entsprechende Anforderung an den SCM des entfernten Systems. Dieser Mechanismus wird in DCOM durch das Interface IActivation realisiert, welches lediglich die Methode `IActivation::RemoteActivation` implementiert.

Nachdem jetzt die Instantiierung von DCOM-Komponenten behandelt wurde, ist die Überwachung der Lebenszeit zu betrachten. Der hier verwendete Ansatz baut auf das Reference Counting des IUnknown Interfaces auf (siehe Anhang). Bei verteilten Objekten tritt dabei das Problem auf, daß ein Klient (unbeabsichtigt) terminiert, ohne seine Referenz auf das Interface zu dekrementieren. Dieses Problem wird durch ein periodisches *Ping* gelöst, indem ein ORPC auf die Serverkomponente ausgelöst wird. Der einfache Ansatz, bei dem jeder Klient jedes verwendete Interface regelmäßig kontaktiert, ist allerdings bezüglich der verbrauchten Ressourcen zu teuer. Als Optimierungsansatz werden unter DCOM diese Pings gebündelt und als OXID Resolver übertragen. Dadurch kann festgestellt werden, welche Interfaces zu einer Komponente gehören und welche Komponenten auf einer Maschine laufen. Dieser Ping wird durch den Aufruf von `IObjectExporter::ComplexPing()` ausgeführt. Danach kann ein einfacher Ping von einem Resolver zum anderen mit `IObjectExporter::SimplePing()` übertragen werden. Werden von einem Klienten innerhalb eines vorgegebenen Intervalls keine weiteren Pings erhalten, wird dessen Referenz gelöscht. Dadurch ist eine verteilte Garbage Collection unter DCOM implementiert [CHAP96].

[18] Diese Bezeichnung ist unglücklich, da IObjectExporter durch den OXID Resolver, nicht aber durch den Exporteur selbst implementiert wird. Weiter ist die Namensgebung irreführend, da hier kein COM Interface realisiert ist, sondern eine RPC Schnittstelle, die direkt durch RPC Aufrufe angesprochen wird.

Im DCOM gibt es zwei Kategorien von Sicherheit. Die erste wird als *Aktivierungssicherheit* bezeichnet und behandelt, wie neue Komponenten instantiiert, neue und bestehende Komponenten verbunden, und wie bestimmte Dienste, wie zum Beispiel die ROT, gesichert werden. Die zweite wird als *Aufrufsicherheit* bezeichnet, und stellt Sicherheit auf Aufrufebene zwischen Komponenten zur Verfügung, und etabliert sichere Verbindungen zwischen Klienten und Komponentenserver. Aspekte der Sicherungs-API sind inhärent plattformabhängig. Die MS Windows Versionen werden als Referenz angegeben [DRAFT961].

Komponenten werden durch Klienten entweder durch statische, persistente Einträge in der Registry mit Verweis auf den Servercode für den SCM, oder durch dynamische Veröffentlichung via `CoRegisterClassObject()` beziehungsweise Registrierung in der ROT adressiert. Aktivierungssicherheit hat damit also einen statischen und einen dynamischen Aspekt. Sie wird automatisch auf dem SCM einer Maschine angewendet und überprüft Sicherheitsinformationen anhand der eigenen Registry oder dynamisch gesammelter Komponenten in internen Tabellen. Die Berechtigung eines Objektes (s. Abb. 8: Die Verantwortung eines Objektes.) wird durch ACL geregelt.

Aufrufsicherheit tritt als zweite Instanz nach der Aktivierung der Komponente in Kraft. Verschiedene Sicherungsdienste können Anwendung finden, zu den wichtigsten gehören

- Authentifizierung
- Autorisation
- Datenintegrität
- Datensicherheit

Unabhängig von den zugrundeliegenden Mechanismen ist die Aufrufsicherheit von DCOM durch zwei Grundoptionen definiert: *Automatische Sicherheit* und *Interface Sicherheit*. Bei der ersten kann eine Komponente eine Sicherheitsstufe vorgeben, die für alle Methodenaufrufe gültig ist. Bei der Interface Sicherheit können verschiedene Sicherheitsstufen für die einzelnen Interfaces vorgegeben werden. Eine gleichzeitige Anwendung beider Mechanismen ist möglich [CHAP96].

Die folgende Tabelle faßt die Merkmale des DCOM zusammen.

Eigenschaft	***DCOM***
Parameter Marshalling	ja
Parameterübergabe	in, out, in/out
Laden von dynamischen Stubs / Klassen	nein
Garbage Collection	ja
dynamic discovery / dynamic invocation	ja
Sicherheit auf Transportebene	ja
Sicherheit auf Transaktionsebene	ja (explizit)
Persistente Namensgebung	ja (Moniker)
Persistente Objektreferenzen	ja
Sprachneutrales Transportprotokoll	ja
Zugangskontrolle	ja
Einsatz durch Firewalls	ja
proprietär	nein

Tab. 11: Eigenschaften von DCOM [ORF97].

Bemerkenswert ist dabei vor allem, daß persistente Objektreferenzen nicht explizit von dem Modell unterstützt werden. Implizite Unterstützung bietet aber die Verwendung von Moniker für die Speicherung der Objektreferenzen.

Für die Konfiguration von DCOM auf den einzelnen, beteiligten Maschinen, stellt Microsoft keine standardisierten Werkzeuge zur Verfügung. Die GUID der Serverkomponente ist automatisch in die Tabelle der Moniker sowohl auf Klienten- als auch auf Serverseite eingetragen

worden. Beispielhaft wird hier die Konfiguration des in Kapitel 4.3 implementierten Servers für den Benchmarkversuch vorgestellt:

Application Name:	DCOMServer.clsDCOMServer
Application Type:	Local Server
Local Path:	C:\WINNT\system32\DCOMServer.exe

Auf der Klientenseite kann nun der Ausführungsort entsprechend dem Standort des Servers geändert und die Sicherheitseinstellungen nach Bedarf vorgenommen werden.

Mögliche Sicherheitseinstellungen sind:

- Zugriffsberechtigung
- Startberechtigung
- Konfigurationsberechtigung

Standardeigenschaften der DCOM Konfiguration werden immer auf dem Server eingestellt. Dabei sind folgende Konfigurationen möglich:

- Standard-Authentifizierungsebene (nach Sicherheitsebene aufsteigend):
  Legt die Sicherheit für Pakete bei der Kommunikation zwischen Anwendungen fest.

  [Kein — Keine Sicherheitsüberprüfung bei der Kommunikation zwischen Anwendungen.

  [Standard — Die Sicherheitsebene wird für den installierten Echtheitsbestätigungsdienst auf die Standardeinstellung gesetzt. Die Standardeinstellung von WinNT ist *Verbinden*.

  [Verbinden — Die Sicherheitsüberprüfung wird nur für die Erstverbindung durchgeführt.

  [Anrufen — Die Sicherheitsüberprüfung wird bei jedem Anruf während der gesamten Verbindung durchgeführt.

  [Paket — Die Identität des Senders wird verschlüsselt, um die Echtheit des Senders zu gewährleisten.

  [Paketintegrität — Die Identität und Unterschrift des Senders werden verschlüsselt, um die Echtheit des Senders zu gewährleisten und um sicherzustellen, daß die Pakete während der Übertragung nicht verändert werden.

  [Packet Privacy — Das gesamte Paket, einschließlich der Daten, sowie die Identität und die Unterschrift des Senders werden in der höchsten Sicherheitsebene verschlüsselt.

- Standard-Verkörperung (nach Sicherheitsebene aufsteigend):
  Die Ebene der Berechtigung, die eine Klientenanwendung einer Serveranwendung für die Bearbeitung von Tasks für die Klientenanwendung zuweist.

  [Anonym — Die Serveranwendung bearbeitet Tasks für den Klienten, ohne die Identität der Klientenanwendung zu kennen.

  [Identität — Die Serveranwendung kann die Identität der Klientenanwendung überprüfen.

  [Impersonate — Die Serveranwendung kann die Identität der Klientenanwendung nur annehmen, indem sie die Tasks als Klientenanwendung bearbeitet. Die Serveranwendung nimmt die Identität der Klientenanwendung nur auf dem Computer an, auf dem die Serveranwendung ausgeführt wird.

[Delegate — Die Serveranwendung kann Tasks auf einem anderen Computer als Klientenanwendung bearbeiten. Die Serveranwendung kann die Identität der Klientenanwendung auf dem Computer, auf dem die Serveranwendung ausgeführt wird, oder auf einem beliebigen anderen Computer annehmen (Der Echtheitsbestätigungsdienst von WinNT Servern unterstützt nicht *Delegate*).

- Zusätzliche Sicherheit für Referenz-Protokollierung:
  Diese Option legt fest, daß die Serveranwendung die Anzahl der verbundenen Klientenanwendungen verfolgt. Dadurch wird sichergestellt, daß ein Serverprozeß nicht durch eine Klientenanwendung unterbrochen wird, indem die Referenznummer künstlich auf Null gesetzt wird.

Für Modifikationen an dieser Konfiguration sind administrative Rechte auf der Workstation des jeweiligen Servers erforderlich. Ein Konfiguration aus Anwendungen heraus (insbesondere durch ActiveX-Komponenten) ist aus Sicherheitsgründen nicht möglich.

Da DCOM ein inhärent sichereres Modell ist, kann es ohne die Kapselung eines VPN eingesetzt werden: DCOM kann damit einfache TCP/IP Netzwerke verwenden. Für den Einsatz durch Firewalls bieten sich folgende Möglichkeiten [MS965]:

- DCOM verwendet einen einzelnen Port für die Initialisierung von Verbindungen und weist eine konfigurierbare Menge von Ports der tatsächlich auf einer Maschine laufenden Komponente zu (ein Port pro Prozeß).
- Application-level Proxies können einfach erstellt werden. Diese Möglichkeit kann entweder generisch (durch Weiterleitung konfigurierbarer DCOM Aktivierungen und Methodenaufrufe) oder anwendungsspezifisch entwickelt werden.
- DCOM kann, wenn unbedingt nötig, auch durch HTTP getunnelt werden. Dadurch können alle Einschränkungen von Firewalls umgangen werden, wenn nicht aktive Filter verwendet werden.

Diese Optionen bieten die sichere Verbindung von DCOM-Komponenten über die Grenzen eines Intranets hinaus an. Sowohl protocol-level als auch application-level Gateways können überbrückt werden. Durch den Verzicht auf die Verwendung von VPN, eignet sich das Modell ausgesprochen gut für den kombinierten Einsatz im Intra- und Internet.

3.2.4 Vergleich von COM und CORBA

3.2.4.1 Übersicht

COM und CORBA bieten Kommunikation im Klient/Server-Modell an. Um einen Dienst eines Servers in Anspruch zu nehmen, wird vom Klienten eine Methode auf dem entfernten Objekt aufgerufen, das als Server im Klient/Server-Modell agiert. Der Dienst des Servers ist in einem Objekt über eine Schnittstelle (bzw. ein Interface) implementiert, das konform zu einer (M)IDL ist. Die IDL ist also in beiden Systemen der grundlegende „Vertrag“ zwischen Klient und Server. CORBA unterstützt dabei, im Gegensatz zu COM, neben Verschachtelung, Polymorphismus und einfacher Vererbung auch mehrfache Vererbungen von Interfacedefinitionen und Objekten. An dieser Stelle wird im COM ein Objekt mit mehrfachen Schnittstellen spezifiziert um ein analoges Verhalten zur Mehrfachvererbung sowohl auf der IDL- als auch auf der Implementierungsebene zu ermöglichen. Auf die Mehrfachvererbung in COM muß verzichtet werden, um eine Kompatibilität im Binärcode zu gewährleisten. Diese ermöglicht eine Versionskontrolle von Binärkomponenten.

CORBA ist ein zu COM konkurrierender Standard für verteilte Objekte. Er definiert ein abstraktes Objektmodell, das Komponenten und deren Schnittstellen beschreibt. Weiter liefert er eine standardisierte Abbildung von abstrakten Objektdefinitionen auf konkrete Programmiersprachen. Ein binärer Standard wird dabei nicht definiert. Verschiedene ORB Implementierungen,

die sich an den CORBA-Standard halten, können Kompatibilität auf Quellcodeebene, nicht aber Austauschbarkeit von binären Objekten erreichen. Dies ist einer der Hauptgründe, warum es keinen Markt für wiederverwendbare CORBA-Objekte gibt. Hersteller von Komponenten müssen den Quellcode veröffentlichen und diesen auf allen Zielplattformen und ORB Implementierungen testen. Ein Schutz des intellektuellen Eigentums kann so nicht geboten werden.

CORBA definiert weiter einen Standard für Inter-ORB Kommunikation, der zwei kompatiblen ORB Implementierungen erlaubt, gegenseitig Objektmethoden aufzurufen. ORB Implementierungen wie DSOM von IBM oder ORBIX von IONA stellen proprietäre Erweiterungen zum Objektmodell, zur Sprachenbindung und zum Inter-ORB-Protokoll zur Verfügung. Um die volle Leistungsfähigkeit einer Plattform ausschöpfen zu können, muß man die ORB-übergreifende Interoperabilität opfern. Ein Ausweg bildet das durch CORBA definiert IIOP, das auf die Verwendung im Internet zielt. Es bildet quasi eine Schnittmenge der Inter-ORB-Protokolle [MS965].

Die Objektinteraktion beruht bei beiden Systemen auf einem objektorientiertem RPC-Mechanismus. In COM wird der Klienten-Sockel als *proxy* und der Server-Sockel als *stub* bezeichnet. Bei CORBA wird der Klienten-Sockel als *stub* und der Server-Sockel als *Skeleton* bezeichnet. Hier bezieht sich der Terminus *proxy* auf ein instantiiertes Skeleton.

Eine Darstellung der Unterschiede soll anhand von Middleware-Schichten, in der die RPC Struktur beider Systeme unterteilt werden kann, erfolgen.

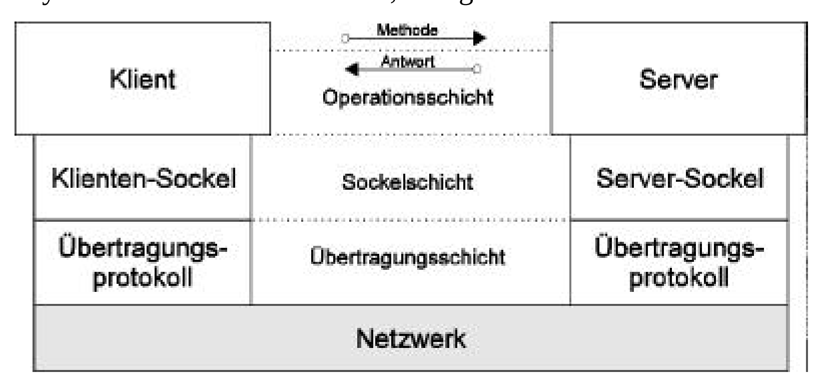


Abb. 27: Basismodell eines RPC Systems.

3.2.4.2 Operationsschicht

Die Operationsschicht ist die Programmarchitektur, die als einzige für den Entwickler sichtbar ist. Der Hauptunterschied zwischen COM und CORBA liegt hier in der Form, wie ein Klient eine Schnittstelle spezifiziert, also den „Class Factories" von CORBA und dem IUnknown Interface von COM. Im COM kann für die Instantiierung eines Objektes zunächst entweder ein Moniker oder die ROT verwendet werden. Optional können darüber hinaus „Class Factories" verwendet werden. Ein Serverobjekt kann `CoRegisterClassObjekt()` aufrufen, um einen Interfacezeiger zu veröffentlichen, der wiederum durch den Aufruf von `CoGetClassObject()` vom Klienten abgerufen werden kann. Wichtig dabei ist, daß der Aufruf von `CoCreateInstance()` nicht zwangsläufig ein neues Objekt instantiiert. In der Methode `IClassFactory::CreateInstance` kann ein Server festlegen, daß immer der gleiche Interfacezeiger zurückgegeben wird. So können unterschiedliche Klienten auf die gleiche Objektinstanz in einem bestimmten Zustand zugreifen [CHU97].

In CORBA wird ein Objekt durch Aufruf einer Methode auf einer bestehenden Objektreferenz aktiviert. Einige Implementierungen bieten, wie auch zum Beispiel JavaIDL, eine explizite `bind()` (org.omg.CosNaming.Binding) Operation an, um ein Serverobjekt zu aktivieren und dessen Referenz zu erhalten. Der Klient kann eine bestehende Instanz eines Objektes oder ein neues Objekt referenzieren. Zu jeder Zeit kann der Klient eine Objektreferenz durch

`objekt_to_string()` speichern und später wieder durch `string_to_objekt()` zurückkonvertieren.

Ein weiterer interessanter Unterschied liegt in der Fehlerbehandlung. CORBA bietet eine Unterstützung für C++ Ausnahmen, die um einige CORBA-spezifischen Ausnahmen erweitert sind. Darüber hinaus können benutzerdefinierte Ausnahmen in der IDL eingeführt werden. COM verlangt auf dieser Ebene von jeder Methode einen 32-bit Fehlercode (HRESULT). Auf Sprachenebene bietet das System einen Dienst, das IErrorInfo Objekt, um Fehler im HRESULT abzufangen. Das (D)COM Übertragungsprotokoll bietet den Mechanismus *body extensions*, der die Übertragung einer breiten Fehlerinformation, zum Teil auch im Klartext, ermöglicht.

Unterschiede ergeben sich auch im zeitlichen Ablauf der Operationen, die in [CHU97] präzise abgebildet sind und verglichen werden.

3.2.4.3 Sockelschicht

Die Sockelschicht ist für die Zeiger auf Interfaces und Objektreferenzen auf verschiedene Prozesse verteilt. Sie bietet also Klienten und Server die Illusion eines gemeinsamen Adreßraums. Hier liegt der Unterschied in den Mechanismen, mit denen Serverobjekte registriert und dem Zeitpunkt, zu dem Sockel instantiiert werden. Das hier behandelte Verfahren des Marshalling unter (D)COM wird als *standard marshalling* bezeichnet. Ein *custom marshalling* wird alternativ angeboten. Durch die Implementierung des IMarshal Interfaces übernimmt der Server die Kontrolle über das verwendete Marshalling-Verfahren und die Art, wie der Klient mit dem Server kommuniziert. Diese alternative Option ist sehr ausbaufähig [CHU97].

CORBA definiert den OA für die Anbindung der Objektimplementierung an den ORB. Dabei werden für unterschiedliche Objekttypen entsprechende OA zur Verfügung gestellt. Für Standardimplementierungen steht der BOA zur Verfügung.

Auch hier finden sich Unterschiede im zeitlichen Ablauf der Operationen, die in [CHU97] aufgeführt sind. Beide Methoden führen aber zum gleichen Ziel.

3.2.4.4 Übertragungsschicht

Die Übertragungsschicht erweitert die Sockelschicht auf eine Verteilung auf verschiedene Maschinen. Der Unterschied auf dieser Ebene zeigt sich vor allem in der Methode, wie entfernte Interfacezeiger oder Objektreferenzen dargestellt werden, um die Server Endpointinformation zum Klienten zu übertragen, und in den Darstellungsformaten, in denen Daten über heterogene Systeme übertragen werden.

CORBA spezifiziert kein Protokoll für die Kommunikation zwischen Klienten und Objektserver über einen ORB des gleichen Herstellers. Das Protokoll für ORB-übergreifende Kommunikation ist vom Hersteller der ORB abhängig. Für die Interoperabilität zwischen verschiedenen ORB-Produkten ist ein „General Inter-ORB Protocol (GIOP)“ spezifiziert. Eine gesonderte Spezifikation ist für die Abbildung von GIOP auf TCP/IP Verbindungen im „Internet Inter-ORB Protocol (IIOP)“ festgelegt.

Die (D)COM Übertragungsschicht basiert auf MS RPC. Es bietet im Unterschied zu CORBA eine „verteilte Garbage Collection“ durch ein *pinging protocol*.

3.2.4.5 Zusammenfassung

COM und CORBA bieten, auf einer ähnlichen Grundarchitektur, eine Infrastruktur für verteilte Objekte. Die Hauptunterschiede liegen darin, daß erstens COM Objekte mit mehreren Interfaces unterstützt, die mit `QueryInterface()` traversiert werden können. Dadurch ergibt sich die Möglichkeit, daß ein Objektsockel mehrere Interface-Sockel im entfernten Adreßraum dynamisch laden kann. Dieses Konzept besteht unter CORBA nicht. Zweitens unterstützt CORBA Implementierungsvererbung. COM verzichtet, für die Erhaltung seiner Standards auf

Binärcodeebene, auf dieses Merkmal. Jede CORBA Schnittstelle wird vom `CORBA::Object` abgeleitet. Dies ist der Basiskonstruktor, der grundlegende Aufgaben wie Objektregistrierung, Referenzengenerierung, Skeleton Instantiierung usw. bereitstellt. In COM werden diese Aufgaben entweder explizit durch den Server oder dynamisch von der COM-Laufzeitumgebung übernommen. Drittens ist das Übertragungsprotokoll von (D)COM auf MS RPC ausgelegt [CHU97].

Die folgenden Tabelle faßt die begrifflichen Unterschiede der Modelle vergleichend zusammen.

	COM	***CORBA***
Operationsschicht		
Allgemeine Basisklasse	IUnknown	CORBA::Object
Objektreferenz	CLSID	interface name
Schnittstellenreferenz	IID	interface name
Objektaktivierung durch Klient	CoCreateInstance()	bind()
Object handle	Interfacezeiger	Objektreferenz
Sockelschicht		
Name der Implementierungsabbildung	Registry	Implementation Rep.
Implementierungsabbildung	Type Library	Interface Repository
Typeninformation für Methoden	SCM	ORB
Lokalisierung	SCM	OA
Klientensockel	proxy	stub/proxy
Serversockel	stub	Skeleton
Übertragungsprotokoll		
Objektendpunkt Auflösung	OXID resolver	ORB
Erstellung von Objektreferenzen	object exporter	OA
Objektendpunkt	object exporter	OA
Objektreferenz	OBJREF	IRO o. Objetctref.
Marshalling Format	NDR	CDR
Bezeichner für Schnittstelleninstanz	IPID	object_key

Tab. 12: Zusammenfassung von Bezeichnungen und Entitäten in COM u. CORBA [CHU97].

4 Beispielentwicklung und -implementierung

4.1 Überblick

Die in dieser Arbeit vorgestellten Beispiele sollen die Vor- und Nachteile des Einsatzes von COM und CORBA als Middleware auf Basis eines (langsamen) Intranets und des Internets aufzeigen. Dabei wird jeweils eine Anwendung implementiert, die einen „Active View“ auf einen Datenbestand liefert. Ein Klient mit GUI soll dem Anwender einen Ausschnitt aus einem Datenbestand grafisch und ergonomisch darstellen. Die Darstellung der einzelnen Datensätze soll, nach Möglichkeit, den aktuellen Datenbestand widerspiegeln und Änderungen sofort dem Klienten anzeigen. Transaktionssicherheit soll bei allen Operationen gewährleistet sein.

Nachdem die Middlewareplattformen im vorherigen Kapitel ausführlich diskutiert wurden, wird hier ein Augenmerk auf die Internetprogrammiersprachen selbst gerichtet. Auf der einen Seite hat sich JAVA klar als Internetprogrammiersprache, besonders für kleinere Anwendungen etabliert. Diese objektorientierte Programmiersprache der Firma SUN ist gerade aufgrund ihrer Plattformunabhängigkeit für den Einsatz im Internet prädestiniert. Als Middlewarebasis bietet sich hier CORBA gegenüber COM an.

Im Vergleich zwischen den verschiedenen Middlewareplattformen wird auf der anderen Seite COM als Konkurrenzmodell für CORBA betrachtet. Als Modell für binäre Komponenten ist bei COM die verwendete Programmiersprache im Grunde unerheblich. ActiveX-Komponenten können, als reduzierte COM Spezifikation, zur Zeit in MS C++, MS Visual JAVA und MS Visual Basic realisiert werden. Exemplarisch wird in dieser Implementierung auf Visual Basic der Firma Microsoft zurückgegriffen. Visual Basic ist eine der am meisten verwendeten Programmiersprachen und ist aufgrund des hohen Abstraktionsgrades leicht zu implementieren.

Attribut	***JAVA***	***ActiveX***
Programmiersprache	Ja	Nein
Sprachneutrale Komponentenarchitektur	Nein	Ja
Virtual Machine	Ja	Nein
Einfach	Ja	Ja
Robust	Ja	Ja
Objektorientiert	Ja	Ja
Plattformunabhängig	Ja	Ja (?)
Multi-Threaded	Ja	Ja
Verteilt	Nein	Ja

Tab. 13: Sind JAVA und ActiveX vergleichbar?

In der obigen Tabelle werden die Unterschiede und Gemeinsamkeiten von JAVA und ActiveX zum besseren Verständnis dargestellt. Es wird deutlich, daß JAVA als Programmiersprache nur in Verbindung mit einer Middleware eine Basis für verteilte Objekte bieten kann. ActiveX basiert dagegen auf dem Komponentenmodell COM, daß mit seiner DCOM Erweiterung die Basis für Verteilung bietet. Binäre ActiveX-Komponenten werden durch Hochsprachen realisiert.

Die Frage nach der Plattformunabhängigkeit von ActiveX ist nicht trivial. Zum einen bietet ActiveX einen offenen Binärstandard für Komponenten, die auf unterschiedlichen Plattformen lauffähig sind. Zum anderen fußen ActiveX-Komponenten auf der Win32-API, die Microsoft-proprietär ist. In der aktuellen Entwicklung finden allerdings Portierungen auf eine Vielzahl von Microsoft-fremden Systemen statt, so daß im folgenden auch ActiveX als eingeschränkt plattformunabhängig bezeichnet wird.

Es soll an dieser Stelle nochmals klargestellt werden, daß eine Implementierung der ActiveX-Komponenten auch in JAVA (genauer MS J++) möglich wäre; dies würde aber der Diskussion

keinen Beitrag leisten. Dagegen kann die Wahl der Programmiersprache MS Visual Basic die Einfachheit der Implementierung verdeutlichen und Vorurteile gegen diese Programmiersprache entkräften. Nicht zuletzt spricht die weite Verbreitung der Sprache für diese Wahl.

So ist es zum Beispiel möglich mit MS J++ entwickelten ActiveX-Komponenten direkt auf „ActiveX Data Objects (ADO)“ zuzugreifen. ADO ist ein „ActiveX Mantel“, der die Datenzugriffstechnologien „Remote Data Object (RDO)“ und „Data Access Object (DAO)“ umschließt, die im weiteren noch vorgestellt werden. Zusätzlich gibt es eine ADO Implementierung in MS J++, die auf „JAVA Beans“ [SUN397] basiert. Dadurch stehen dem Entwickler für Win32-Plattformen zwei mächtige Werkzeuge für die Implementierung von vielschichtigen Datenzugriffsanwendungen zur Verfügung [MSDN997].

4.2 JAVA und JavaIDL

4.2.1 JAVA im Internet und Intranet

Die objektorientierte Programmiersprache *JAVA* bietet die Möglichkeit, sogenannte Applets über dynamische Browser von einem WWW-Server herunterzuladen und dann lokal, d.h. systemunabhängig auszuführen. JAVA ist dabei einfach, multithreaded, garbage-collected, sicher, robust, architekturneutral, übertragbar und dynamisch. Die Programmiersprache ist weitgehend an C++ angelehnt. JAVA-Programme werden in Bytecode transformiert, der ohne weitere Kompilation auf vielen verschiedenen Computern ausgeführt werden kann, sofern es jeweils eine JAVA VM für das entsprechende Betriebssystem gibt.

Das einzigartige an JAVA ist dabei vor allem die resultierende Mobilität und Flexibilität im Einsatz. Es muß lediglich eine JAVA Virtual Machine entwickelt werden, die in einem relativ begrenzten Code (ca. 200 KByte) implementiert werden kann.

Bytecode ist eine Liste von Instruktionen, der optische Ähnlichkeiten mit Maschinencode hat, aber unabhängig vom ausführenden Prozessor ist. Dieser maschinenunabhängige Bytecode wird bei der Ausführung von einem Interpreter, der JAVA VM, abgearbeitet. JAVA wird deshalb gelegentlich nicht nur als Programmiersprache, sondern als neue Plattform bezeichnet. JAVA VM sind derzeit für viele Betriebssysteme und Prozessorplattformen verfügbar und weitere befinden sich in der Entwicklung (vgl. http://java.sun.com/products/index.html).

Bei JAVA-Programmen unterscheidet man zwischen Applets und Applikationen. JAVA-Applets sind JAVA-Programme, die in eine HTML-Seite eingebunden sind und über dieses konfiguriert werden können [MOR97]. Der Browser übernimmt dabei die Funktion der VM. Wenn ein Benutzer auf diese Seiten zugreift, werden die darin verknüpften Klassen des Applets auf den eigenen Computer geladen und über die VM ausgeführt. Im Gegensatz zu der CGI-Programmierung, bei der die Verarbeitung auf dem Server stattfindet, erfolgt die Verarbeitung auf der Seite des Klienten in einem JAVA kompatiblen Browser [DOS96]. Hier kommt vor allem der Sicherheitsaspekt zum tragen. Um nicht unerwünschte Aktionen auf dem Klientensystem ausführen zu können, unterliegen Applets starken Einschränkungen, auf die im weiteren noch eingegangen wird. Im Unterschied zu den Applets, können JAVA Applikationen in der VM des JDK direkt, damit ohne Browser, ausgeführt werden. Sicherheitsrestriktionen können somit hier entfallen [MOR97].

1990 begann James Gosling mit der Entwurfsphase dieser neuen Programmiersprache, die auf die Bedürfnisse der Konsumelektronik besser zugeschnitten sein sollte als die traditionellen Programmiersprachen wie COBOL, C und C++. Das Ergebnis war JAVA, eine sehr schnelle, kleine, verläßliche Sprache, die auf einer Vielzahl von Maschinenplattformen lauffähig ist.

Um 1993 wurde das Internet als textbasiertes Informationssystem durch das WWW um eine GUI erweitert. Das mittlerweile vergrößerte JAVA Team entdeckte, daß eine plattformunabhängige Programmiersprache wie JAVA die ideale Sprache wäre, um Anwendungen für das Internet zu

programmieren. Das Ergebnis war der erste WWW-Browser namens WebRunner, der vollständig in JAVA geschrieben war und später, aus Gründen des Urheberrechts, in HotJava umbenannt wurde (vgl. http://java.sun.com/products/hotjava/1.1/index.htm). Die öffentliche Bekanntgabe der JAVA-Technologie fand auf der SUN Microsystems World Konferenz in San Francisco im Mai 1995 statt. Dort gab Netscape bekannt, daß auch Navigator 2.0, der am meisten verbreitete Browser, JAVA unterstützen wird.

Das JAVA Entwickler Team umfaßt mittlerweile knapp 20 Personen und arbeitet weiter an der Verbesserung der Sprache.

4.2.1.1 Eigenschaften

Die Eigenschaften von JAVA und die damit verbundenen Vorteile gegenüber anderen objektorientierten Programmiersprachen wie z.B. C++ und Visual Basic (auf die Objektorientierung von Visual Basic wird im folgenden noch eingegangen) lassen sich wie folgt zusammenfassen.

JAVA ist einfach und klein. JAVA stammt in seinen Ursprüngen von C und C++ ab. Der objektorientierte Ansatz und ein Großteil der JAVA Syntax sind an C++ angelehnt. Zwangsläufig ist JAVA für Entwickler von C und C++ Programmen leicht zu erlernen. Bei der Entwicklung von JAVA wurde bewußt auf die Einbeziehung von komplizierten Aspekten aus C++, wie Zeiger und Speicherverwaltung, verzichtet. Die Speicherverwaltung und „Garbage Collection“ werden automatisch von JAVA ausgeführt. Abschließend ist JAVA in bezug auf Basistypen sehr einfach und streng strukturiert. Zur Manipulation von Variablen ist fast immer ein expliziter Cast notwendig, was Fehler im Umgang mit Variablen vermindert [MOR97].

JAVA ist objektorientiert. Objetkorientierung ist bei Programmiersprachen mittlerweile zum Standard geworden. Dabei wird nicht auf Prozeduren und Daten, sondern auf Zustände, Aktivitäten und Kommunikation von Objekten aufgebaut. Diese Objekte verhalten sind nach außen hin einheitlich und sind gleichberechtigt, besitzen Zustände, führen Methoden aus und können Nachrichten empfangen und verschicken [HER93]. JAVA hat gegenüber anderen objektorientierte Programmiersprachen, besonders bezüglich Internetprogrammierung und Entwicklung verteilter Anwendungen, Vorteile. Die JAVA-Klassenbibliotheken können zur Laufzeit entweder lokal oder über das Internet „verteilt“ angesprochen werden. Umfangreiche Klassenbibliotheken ermöglichen die Kommunikation über Sockets, was JAVA für den Einsatz im Internet prädestiniert.

JAVA ist inhärent sicher. JAVA bietet Sicherheit auf verschiedenen Ebenen. Erstens ist es mit der Sprache an sich sehr schwierig gefährliche Anwendungen zu entwickeln. Die Vermeidung von Zeigern trägt einen wichtigen Teil zu dieser Eigenschaft bei, da Eingriffe in den Speicherbereich anderer Anwendungen nicht ermöglicht werden. Ein weiterer Sicherheitsaspekt ist der „*Bytecode Verifier*“. JAVA Quellcode wird zunächst in Bytecode umgewandelt. Vor der Laufzeit wird dieser Bytecode von dem Verifier auf unzulässiges Verhalten hin überprüft. Weiter sind besonders Applets, also JAVA-Anwendungen, die über das Internet innerhalb von Browsern ausgeführt werden, eingeschränkt. Applets können nicht auf das Dateisystem des Klienten zugreifen, neue Fenster, die von JAVA geöffnet werden, tragen ein entsprechendes Logo, um sie (z.B. als Trojanisches Pferd [BRU96]) identifizieren zu können, und Socket-Verbindungen können nur zum WWW-Server, also zu dem Ursprungssystem des Applets aufgebaut werden. Der Verzicht auf Zeiger verhindert weiter Manipulationen am Speicherbereich außerhalb der VM.

JAVA ist plattformunabhängig. Plattformunabhängigkeit ist die Fähigkeit eines Programmes auf unterschiedlichen Betriebssystemen ausführbar zu sein. Insbesondere ist es dafür notwendig, daß Variablentypen auf allen Systemen gleich vereinbart werden und die gleiche Größe besitzen. Ermöglicht wird diese Plattformunabhängigkeit der Programmiersprache durch den von der Plattform abhängigen Interpreter, der den Bytecode zur Laufzeit in für die Maschine spezifischen Code umwandelt.

JAVA ist multithread-fähig. Zur Laufzeit können verschiedene Programmsegmente gleichzeitig, also nebenläufig ausgeführt werden. Die dafür benötigten Hilfsmittel, zum Beispiel zur Synchronisation der Threads, werden von JAVA zur Verfügung gestellt.

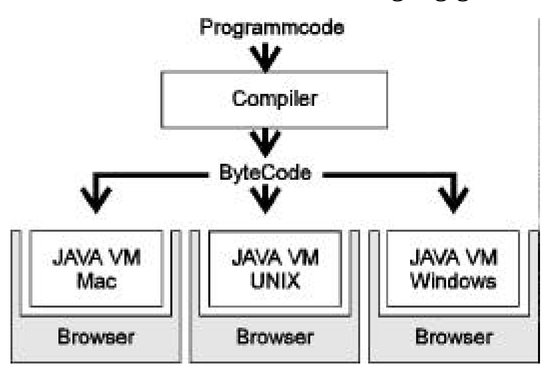


Abb. 28: JAVA Ausführungsmodel.

Viele der Eigenschaften von JAVA sind nicht neu, sondern wurden für andere Sprachen, wie Lisp, Smalltalk, Pascal, Objective-C, Self und Beta entwickelt. Weitere vorteilhafte Eigenschaften sind die Unterstützung von multithreading (die in einer eignen Klasse realisiert, und damit nur bei Bedarf geladen wird) - ein einzelnes JAVA-Programm kann verschiedene Dinge unabhängig voneinander und fortlaufend durchführen, und dadurch hohe Performanz, zum Beispiel für interaktive grafische Applikationen, ermöglichen. JAVA ist sicher, da es Applets keine direkten I/O-Zugriffe auf dem Klientensystem erlaubt; das JAVA-Laufzeitsystem beinhaltet Mechanismen zur Abwehr von Viren. Dabei verwaltet das JAVA-Laufzeitsystem den Speicher selbständig. Klassen werden dann verbunden, wenn sie benötigt werden. Sie können über das Netzwerk geladen werden, d.h. sie müssen nicht auf dem Rechner vorhanden sein, der die VM bereitstellt. Damit ist eine nicht-persistente Softwareverteilung implementiert.

Ein wichtiges Merkmale ist die Speicherbereinigung. JAVA besitzt eine automatische Speicher- und Heap-Verwaltung durch eine „automatic garbage collection“, die als separater Thread realisiert sind. Jedes Objekt, auf das keine Referenz mehr existiert, wird automatisch gelöscht. Der Programmierer braucht sich also nicht, wie zum Beispiel bei C++, mit der Freigabe von Speicherbereichen zu beschäftigen.

Zu den wichtigsten Nachteilen zählt die verringerte Performanz, da JAVA Quellcode vom Interpreter (der die Plattformunabhängigkeit erst möglich macht) zur Laufzeit interpretiert wird. Momentan sind JAVA Applikationen ca. 20-mal langsamer als vergleichbare Anwendungen, die in C++ geschrieben wurden [MOR97]. Weiter beinhaltet JAVA keine Möglichkeit für template classes, wie zum Beispiel C++. Damit kann ein Objekt keine Daten eines Datentyps speichern.

Letztlich ist JAVA, zum Beispiel im Vergleich zu Visual Basic, eine sehr komplizierte Sprache, die nicht unbedingt zum „Rapid Prototyping“ geeignet ist. Dieser Unterschied wird noch in den Beispielimplementierungen deutlich.

4.2.2 JAVA Sicherheitskonzept

4.2.2.1 Einführung

Es stellt sich die Frage, warum gerade die Ausführung von Applets in Browsern solch starken Einschränkungen unterliegt[19]. Zunächst ist die Konfiguration von flexibler Sicherheit, also einer Abstufung der Berechtigungen, die einem ausführbaren Programm zugeteilt werden, und die Konfiguration dieser Stufen durch den Anwender, schwierig zu realisieren und durchzuführen. Gerade diese Forderung kann nicht für alle Betriebssysteme (z.B. Win3x oder Win95) gelten, auf

[19] Siehe auch „Frequently Asked Questions - Applet Security“ [SUN197].

denen die VM implementiert werden soll. Jedes Applet wird in einer eigenen, gesicherten Umgebung ausgeführt, die man als „Sandkasten“ bezeichnen kann. Die „Wände“ des Sandkastens sind die Sicherheitsregeln, die dem Applet durch die VM auferlegt werden. Dieses Sicherungsmodell wird als „Sandboxing“ bezeichnet [CHAP96]. Nur ein kleines Schlupfloch in der „Wand des Sandkastens“ würde das gesamte Sicherungsmodell der Virtual Machine (VM) zusammenbrechen lassen. Das zweite Problem ist, daß ein Anwender ein Kriterium braucht, um ein bestimmtes Applet einer adäquaten Sicherungsstufe zuzuordnen. Eine, und wohl auch die beste Lösung, ist die Sicherstellung der Herkunft des Programmes oder seines Autors. JavaSoft hat ein entsprechendes, proprietäres Modell für digitale Unterschriften entwickelt, dessen Auslieferung sich aber aufgrund von amerikanischen Exportbestimmungen verzögert [BRU96].

4.2.2.2 Die vier Schichten des JAVA Sicherheitsmodells

Die erste Schicht des Sicherheitsmodells besteht in der inhärenten Sicherheit von JAVA und resultiert aus der Sprache selbst, dem dazugehörigem Compiler und seinem Speicherverwaltungsmodell [BRU96]. Dabei bezieht sich die Sicherheit nicht auf den Schutz vor unseriösen Programmieren, sondern dem Schutz vor fehlerhaften Programmen. Zur Laufzeit wird überprüft, ob Referenzen auf Elemente innerhalb der Feldgrenzen liegen um zu vermeiden, daß auf fremde Speicherbereiche zugegriffen wird. Weiter werden alle Casts auf Sprachenkonformität überprüft. JAVA unterstützt, im Gegensatz zu zum Beispiel C++, keine Zeiger auf bestimmte Speicherbereiche. Der durch JAVA kompilierte Code referenziert Speicher über symbolische Links, die erst bei der Ausführung vom JAVA Interpreter in echte Speicheradressen umgewandelt werden. Der Programmierer kann also nicht mit dem Speicher „arbeiten“, da dessen Verwaltung ganz vom darunterliegenden Laufzeitsystem kontrolliert wird.

In der zweiten Schicht analysiert der *Verifier* den Bytecode, bevor dieser ausgeführt wird. Er stellt insbesondere sicher, daß Zugriffsbeschränkungen nicht verletzt werden, Methodenaufrufe nur mit der korrekten Anzahl von Argumenten ausgeführt werden, die Parameter für den Aufruf den korrekten Typ haben und daß ausschließlich korrekte Typenumwandlungen durchgeführt werden. Der Bytecode Verifier macht dabei keine Annahmen über die ursprüngliche Quelle des Codes und fungiert als eine Art Wächter, der sicherstellt, daß Code, der zur Ausführung an den JAVA Interpreter weitergegeben wird, ohne Bedenken ausgeführt werden kann. Importierter Code kann nicht ausführt werden. Nach der Überprüfung sind folgende wichtige Eigenschaften des Codes bekannt [BRU96]:

- Der Code erzeugt keinen Operandenüber- oder unterlauf.
- Die Typen der Parameter aller Anweisungen im Bytecode sind immer korrekt.
- Keine illegalen Datenkonvertierungen haben stattgefunden.
- Zugriffe auf die Objektfelder sind legal (privat, öffentlich oder geschützt).

Nachdem der Verifier den Bytecode überprüft und freigegeben hat, übergibt er diesen in der dritten Schicht an den *Class Loader*. Dieser unterteilt alle Klassen in lokale und externe Klassen. Erstere, wie zum Beispiel die VM selbst, residieren auf dem lokalen Dateisystem des Klienten, letztere müssen über das Netzwerk geladen werden. Alle lokale Klassen erhalten dabei einen gemeinsamen Namensraum während alle externen Klassen einen getrennten Namensraum zugewiesen bekommen. Dadurch wird verhindert, daß beide Klassentypen im Namensraum Schnittmengen bilden und sich so gegenseitig manipulieren. Durch diese Bevorzugung der lokalen Klassen wird verhindert, daß eine externe Klasse eine eingebaute Klasse „überschreibt“.

Abschließend wird in der vierten Sicherungsschicht durch den *Security Manager* das Dateisystem des Klienten vor Zugriff durch das Applet geschützt, ortsabhängige Behandlung von Klassen unterbunden und die Kommunikation über Socket-Verbindungen überwacht. Die Methoden der Klasse `java.lang.SecurityManager` implementieren dabei ein Berechtigungsprofil. Dieses ist in den aktuellen Versionen des JDK ausgesprochen restriktiv, um Klienten vor möglichen Defiziten im Sicherungsmodell auf jeden Fall zu schützen. Zukünftig soll der Security

Manager auch in kommerziellen Browsern individuell angepaßt werden können. Diese Entwicklung ist aber noch nicht abzusehen.

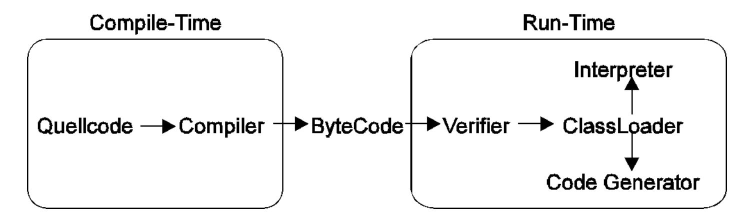


Abb. 29: Verarbeitungsweg von JAVA Quellcode bis zur Ausführung.

Konfigurationsalternativen für den URL-Zugriff bestehen aus

1. No Access — Ein Applet darf nicht auf Objekte über eine URL zugreifen.
2. Applet Host — Ein Applet darf nur Objekte von dem Server laden, von dem es selbst geladen wurde.
3. Firewall — Ein Applet, das innerhalb einer Firewall (also im Intranet) geladen wurde, darf auf alle URLs zugreifen, während alle anderen Applets nur auf URLs außerhalb der Firewall zugreifen dürfen.
4. Unrestricted — Keine Zugriffsbeschränkungen.

Analog können auch die URLs beschränkt werden, von denen JAVA-Applets geladen und ausgeführt werden dürfen. Vertrauenswürdige Systeme können so logisch in das inbound Netzwerk integriert werden, ohne daß sie zum Intranet gehören müssen.

4.2.3 Datenbankanbindung in JAVA

Um eine gemeinsame Basis-API für die Datenbankanbindung zu bieten, hat SUN Microsystems die „JAVA Database Connectivity (JDBC)“ entwickelt. JDBC definiert eine Menge von Schnittstellen, die den datenbankunabhängigen Zugriff auf Daten ermöglichen. Theoretisch kann eine Anwendung unter Verwendung der JDBC-API und SQL-2 Anweisung jede SQL-2 kompatible Datenbank ansprechen, sofern diese einen JDBC-Treiber anbietet. Die Daten können in unterschiedlichsten Formaten und unter Verwendung unterschiedlicher Technologien gespeichert werden. Die meisten Systeme verwenden zur Zeit eines der drei folgenden „Datenbank Management Systeme (DBMS)“ [MOR97]:

- Relationale Datenbanken (RDBMS)
- Objekt-Relationale Datenbanken (OORDBMS)
- Objektorientierte Datenbanken (OODBMS)

Dabei sind die relationalen Datenbanken am häufigsten zu finden. Unabhängig von diesen Systemen sind weitere Unterschiede, wie hierarchische Datenbanken und Dateisysteme zu differenzieren. Jede API, welche die Unterstützung aller möglichen Kombinationen anstrebt, wird zwangsläufig mit einer unzureichenden Funktionalität enden. JDBC stellt jedoch keine besonderen Anforderung an das zugrundeliegende DBMS, sondern spezifiziert lediglich die JDBC-Implementierung selbst. Hauptvorgabe ist dabei die Unterstützung von ANSI-SQL-2 [RIC90]. Diese Sprache ist in DBMS sehr weit verbreitet. Die schwierige Aufgabe, diesen Mantel des JDBC um das bestehende DBMS zu implementieren, liegt bei dem jeweiligen Datenbankhersteller [MOR97].

Das `java.sql.DriverManager` Objekt überwacht dabei die Verbindung zur Datenbank, die mit einer Datenbank-URL (jdbc:msql://*hostname:port*/*database*) angegeben wird. Dieser Manager ist, abgesehen von Ausnahmebehandlung und Unterklassen wie `java.Util.Date`, die einzige instantiierbare Klasse des JDBC.

Jede JDBC-Implementierung muß folgende Schnittstellen anbieten [MOR97]:

- `java.sql.Driver`
  Der *DriverManager* verwendet einen Treiber, um die Datenbank-URL zu verarbeiten. Kann ein Treiber in der angegebenen Liste die Verbindung zur Datenbank aufbauen, wird dieser verwendet [ROD97]. Da Anwendungen Treiber nur indirekt referenzieren, findet diese Schnittstelle kaum direkte Anwendung.
- `java.sql.Connection`
  Eine Connection ist eine einzelne Datenbanksitzung. Sie speichert Informationen zu dieser Sitzung und stellt die Objekte *Statement*, *PreparedStatement* und *CallableStatement* zur Verfügung.
- `java.sql.Statement`
  Ein Statement ist eine SQL-Anweisung für eine Datenbank. Als Ergebnis wird entweder ein ResultSet, oder die Anzahl der veränderten Datensätze zurückgegeben. Unterklassen sind `java.sql.PreparedStatement` und `java.sql.CallableStatement`.
  Ersteres ist ein vorkompilierter Datenbankaufruf, der in einer Stored Procedure (ohne OUT und INOUT Parameter) gespeichert wird. Für *gespeicherte Prozeduren* (~ Stored Procedures) mit OUT oder INOUT Parametern wird die CallableStatement Klasse verwendet.
- `java.sql.ResultSet`
  Das *ResultSet* ist die Ergebnismenge einer SQL-Abfrage. Dieses Objekt ermöglicht einer Anwendung Datenreihen sequentiell auszulesen und zu verarbeiten.
- MetaData Schnittstelle
  Diese Schnittstelle unterteilt sich in `java.sql.ResultSetMetaData` und `java.sql.DatabaseMetaData`. Sie bietet Informationen nicht über die Daten selbst, aber über deren Struktur. Dabei wird zwischen einzelnen Ergebnismengen und der Datenbank selbst unterschieden.

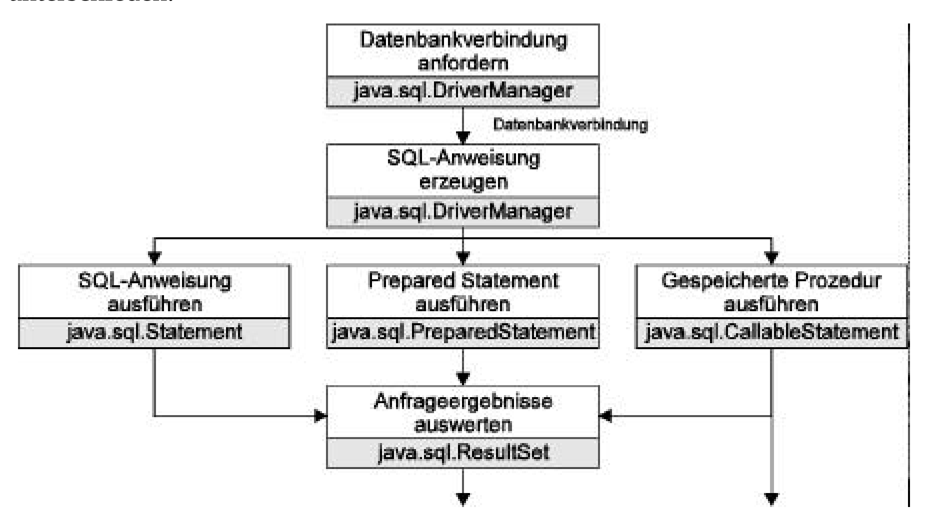


Abb. 30: Die Klassenarchitektur von JDBC [DIC97].

JDBC unterstützt gespeicherte Prozeduren durch Vererbung der Statement-Klasse. Weiter werden Transaktionsmanagement, Cursor-Unterstützung [RIC90] und mehrfache Ergebnismengen unterstützt. Es ist zu beachten, daß JDBC-Treiber, abhängig von der Datenbank, lediglich eine Untermenge dieser Funktionalität anbieten können. Informationen über diese Fähigkeit kann durch die Abfrage von Metadaten der Datenbank erlangt werden. Datenbankoperationen lassen sich mit Hilfe von Methoden einer der Klassen *Statement*, *PreparedStatement* oder *CallableStatement* aufrufen, und deren Ergebnisse mit Hilfe der JDBC-Klasse ResultSet auswerten. CallableStatements bieten dabei die Besonderheit, daß der Umweg über das ResultSet nicht notwendig ist (vgl. Abb. 30) [DIC97]. Dieses Vorgehen würde aber die Vergleichbarkeit zu den Datenobjekten von Mircosoft erschweren und Vergleichsmessungen ergeben ohnehin keinen Vorteil für die direkte Verwendung von Callable Statements [DIC97].

Vorsicht ist bei der Arbeit mit ResultSets geboten. Einige Treiber, unter anderem auch die in dieser Arbeit eingesetzte JDBC-ODBC Brücke, haben Schwierigkeiten bei der Verarbeitung von Daten in einer anderen Reihenfolge, als durch eine Abfrage vorgegeben. Die Abfrage,

```
SELECT name, id FROM test
```

die in dem ResultSet `rs` ausgewertet wird, sollte also nur in der Reihenfolge

```
int    id   = rs.getInt("id");
String name = rs.getString("name");
```

abgearbeitet werden (eine Übersicht der SQL Syntax findet sich z.B. in [JEP97] oder [MSSQL96]). Weiter sollte, in der praktischen Anwendung, jede Spalte nur einmal pro Zeile gelesen werden; bis durch Aufruf der Methode `next()` der nächste Datensatz gelesen wird, sollte nicht wiederholt auf ein Feld des aktuellen Datensatzes zugegriffen werden. Statt dessen sollte jeder gelesene Datensatz sofort einer Variable zugewiesen werden, um dann später verarbeitet zu werden [JEP97].

Ein zweiter, wichtiger Aspekt ist die Mächtigkeit des verwendeten Treibers, die sich aus der Architektur der JDBC-Implementierung ergibt. Mögliche Architekturen für eine JDBC-Implementierung sind:

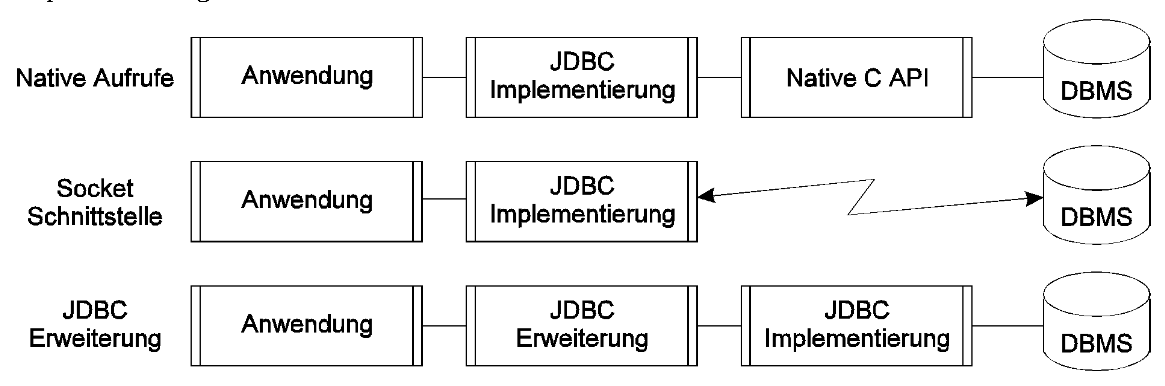


Abb. 31: JDBC Architekturen [MOR97].

Abhängig von den Anforderungen, kann jede dieser Architekturen für eine JDBC-Implementierung geeignet sein. Dabei ist die Erweiterung nicht als echte Eigenentwicklung zu bewerten, da hier kein Code im Sinne der OOP wiederverwendet wird. Mit jeder der drei Formen ist die Anwendung vom tatsächlichen Kommunikationsmechanismus isoliert. Beachtenswert ist noch, daß die Verwendung von nativen C APIs die Plattformunabhängigkeit einschränkt und nicht für den direkten Einsatz unter JAVA geeignet ist. Die VM des Browsers unterbindet native Aufrufe durch Applets.

JDBC basiert, wie auch ODBC, auf dem X/Open „SQL Call Level Interface (CLI)". Damit basiert JDBC nicht nur auf den üblichen APIs auf SQL Ebene, sondern auch auf ODBC. Damit werden Datenbanken, die ODBC unterstützen, ohne Modifikationen auch von JDBC unterstützt. Im Gegensatz zu ODBC, welches eine Unterstützung von asynchroner Verarbeitung anbietet, stellt JDBC Threads zur Verfügung. Dieses Vorgehen führt zum selben Ziel, indem die Fähigkeit von JAVA für asynchrone Verarbeitung durch Threads auf das JDBC ausgedehnt wird. Für jede Abfrage wird ein eigener Prozeß gestartet [LIN96].

Eigenschaft	*JDBC*	*ODBC*
Veröffentlichte Schnittstelle	JAVA Packages	C Funktion Prototypen
Interne Architektur	JAVA Objekte	C Strukturen / C++ Klassen
Objekt Manipulation	JAVA Objekte	Handles
Plattformabhängigkeit	unabhängig	abhängig
Plattformabhängigkeit der Kommunikationsschicht	unabhängig	abhängig
Abhängigkeit von der Datenquelle der Kommunikationsschicht	unabhängig	abhängig
Klientencode	JAVA	C/C++/VB etc.

Tab. 14: Unterschiede von JDBC und ODBC.

Da die große Mehrheit der kommerziellen Datenbanken über eine ODBC-API, kaum aber über eine JDBC-Schnittstelle verfügt, stellt die JDBC-ODBC Brücke einen wichtigen Schritt in der Förderung der Akzeptanz der neuen Programmiersprache JAVA dar. Zudem wäre der Zugriff auf das C-basierte ODBC nur über DLLs möglich, was große Schwierigkeiten bei der Realisierung in JAVA zur Folge hätte. Durch diese Brücke öffnet sich der sehr ausgedehnte Markt der ODBC-fähigen Datenbanken auch für JAVA-Anwendungen [LIN96].

Die folgende Darstellung verdeutlicht die Position der JDBC-ODBC Brücke in einer Datenbank-anwendung unter JAVA.

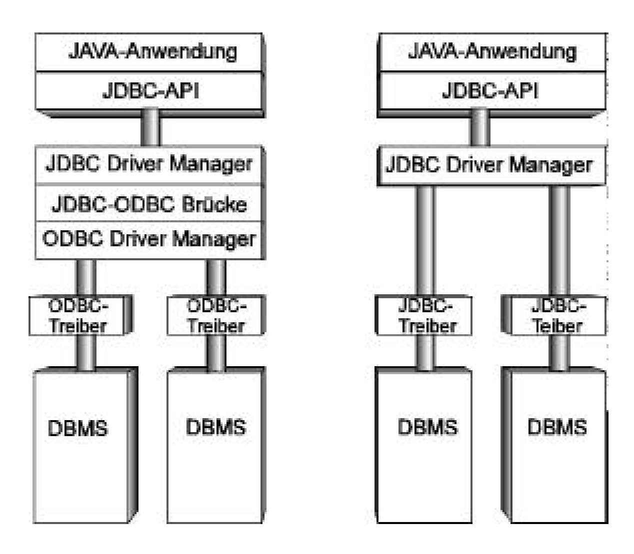


Abb. 32: Integration von ODBC-Treibern in die JDBC-API Schichten.

JDBC besteht aus zwei Hauptschichten: das JDBC-API unterstützt die Kommunikation zwischen der Anwendung und dem JDBC-Treibermanager, die JDBC-Treiber API im JDBC-Treiber-manager unterstützt die Kommunikation zu unterschiedlichen JAVA-Schnittstellen, zum Netz-werk und zu ODBC basierten Treibern. Durch Austausch der JDBC-Treiber können unter-schiedliche Back-end Datenquellen adressiert werden, ohne den Anwendungscode zu ändern. Die JDBC-ODBC[20] Brücke konvertiert dabei JDBC-Aufrufe in ODBC-kompatible Aufrufe auf C Ebene. Aufgrund der gleichen Herkunft von JDBC und ODBC ist die Brücke sehr klein, da nur wenige Umsetzungen notwendig sind. Um so mehr ist es bedauerlich, daß sich Microsoft und SUN Microsystems nicht auf eine einheitliche Spezifikation einigen konnten.

In dieser frühen Testphase der JDBC-ODBC Brücke (alpha2) bildet sie noch nicht die volle Funktionalität von ODBC ab. „Bookmarks“ und „Scrollable Cursors“ werden nicht unterstützt

[20] Die Intersolv JDBC-ODBC Brücke ist ODBC 2.0 kompatibel und auf Win95, WinNT und Solaris ab Version 2.4 verfügbar.

[MSSQL96], was besonders bei Anwendungen einen erhöhten Realisierungsaufwand verursacht, die einen „Active View“ auf Daten zur Verfügung stellen müssen. Genau dies ist aber der Ansatz, der die Nachteile der konventionellen HTML/CGI Realisierung eines Datenbankklienten ausgleicht. Dagegen werden „Metadata“-Klassen als Pendant zur SQLGetInfo() Funktion von ODBC unterstützt [NOR96]. Asynchrone Operationen sind inhärent durch die Thread-Unterstützung von JAVA vorgesehen, die Multithread-Sicherheit muß aber durch den Programmierer gesichert werden. Transaktionen und Rollback werden in Abhängigkeit von dem jeweiligen JDBC-Treiber unterstützt.

Da JDBC praktisch auf ODBC basiert, ist es nicht verwunderlich, daß ein JDBC-Treiber Teile oder die ganze ODBC SQL-Grammatik unterstützt. Dabei wird zwischen drei Graden der Konformität unterschieden, die durch Aufruf der folgenden Methoden differenziert werden:

```
supportsMinumSQLGrammar()
supportsCoreSQLGrammar()
supportsExtendedSQLGrammar()
```

Die hier verwendete Klassifizierung ist analog zu der Definition im Anhang C des „Programmer's Reference for the Microsoft ODBC Software Development Kit“, der mit dem ODBC-SDK von MS V2.10 veröffentlicht wird.

- Minimum
  Die Grammatik, die als Minimalanforderung für die Qualifikation eines DBMS als SQL-DBMS, nicht aber als relationale Datenbank erforderlich ist.
- Kern
  Vereinigt Minimum-Klassifizierung mit einfachen „Data Definition Language (DDL)“- und „Data Control Language (DCL)“-Befehlen. Zusätzliche „Data Manipulation Language (DML)“-Funktionen, Datentypen zusätzlich zu CHAR, SQL Aggregatfunktionen (wie SUM() und AVG()) und eine größere Menge an Ausdrücken für die Selektion von Datensätzen werden hinzugefügt.
- Erweitert
  Beinhaltet Minimum und Kern Grammatik, DML „outer joins“, komplexere DML Ausdrücke als ANSI-SQL Datentypen (zusätzlich long varchar und long varbinary), Batch-SQL Ausdrücke und Prozeduraufrufe.

Eine vollständige Unterstützung wird zum Beispiel durch den MS SQL Server V6.0 angeboten, der in den Implementierungen dieser Arbeit Anwendung findet. Die folgende Tabelle zeigt exemplarisch die grundlegenden SQL-Anweisungen mir ihren jeweiligen Klassifizierungen der Konformität.

SQL-Anweisung	***Konformität***
Alter Table	Kern
Create Index	Kern
Create Table	Minimum
Delete	Minimum
Drop Index	Kern
Drop Table	Minimum
Insert (ein Zeile)	Minimum
Insert (mehrere Zeilen)	Kern
Select (eine Tabelle)	Minimum
Select (mehrere Tabellen)	Kern
Update	Minimum

Tab. 15: Konformitätsklassen von SQL-Anweisungen.

JDBC eignet sich für die Anbindung von Applikationen und Applets an Datenbanken über das Internet oder Intranet, da hier die Klassen in JAVA zur Verfügung stehen. Im Gegensatz dazu,

führt die Anwendung der JDBC-ODBC Brücke durch die Einschränkungen der VM hier zu Behinderungen, da ein Zugriff auf lokale, native Treiber nicht möglich ist. Darüber hinaus ist es Applets nicht möglich, Datenbankserver im Internet zu finden. Damit kann nicht gewährleistet werden, daß ein benötigter Treiber tatsächlich zur Verfügung steht. Letztlich scheitert der Versuch, eine ODBC-Datenquelle durch ein Applet anzusteuern, gänzlich. Da die ODBC-API Software als nativer Dienst implementiert ist, besteht keine Möglichkeit einer direkten Adressierung [LIN96]. Hier kann nur ein 3-Tier-Modell mit einem Datenbankserver realisiert werden. Durch diese Architektur können Applets zum Beispiel über ein ORB auf eine Dienstschicht zugreifen. Diese Dienstschicht kann dann wiederum Aufrufe an den Datenbankserver weitergeben.

Diese starke Einschränkung ließe sich durch direkte Kommunikation mit dem MS SQL Server über Sockets umgehen. Dieses Vorgehen ist allerdings nicht Gegenstand dieser Arbeit und ist mit hohem Aufwand bei der Implementierung verbunden. Als Alternative kann an dieser Stelle auch das Produkt jdbcKona/T3 von WebLogic eingesetzt werden, das einen Proxy für JDBC Verbindungen erstellt. Ähnliche Produkte sind JetConnect von XDB Systems, JDesignerPro von Bulletproof und DataRamp von DataRamp [JEP97].

An dieser Stelle lassen sich, wie in der folgenden Abbildung verdeutlicht, auch die Maschinen trennen. Dies ist für einen Einsatz im Internet ohnehin sinnvoll, da die Dienstschicht auf dem WWW-Server liegen muß, um eine Kommunikation mit dem Applet auf der Seite des Klienten zu ermöglichen. Vom Standpunkt der Firewall ist diese Lokation outbound, potentiell unsicher und für einen Datenbankserver gänzlich ungeeignet. Die Kommunikation zwischen Dienstschicht, also der Servermaschine und dem Datenbankserver kann durch eine Firewall gefiltert und gegebenenfalls auch protokolliert werden. Da sich nur die Servermaschine als Anwender stellvertretend auf dem Datenbankserver authentifizieren muß, kann hier wenigstens ein Minimum an Sicherheit erreicht werden.

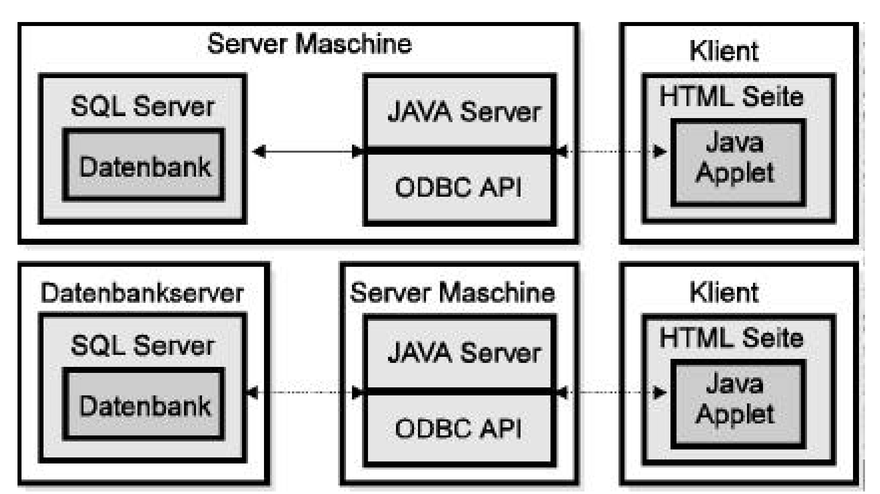


Abb. 33: 3-Tier-Anwendung unter JAVA.

Die JDBC-API ist zur Zeit in der Testversion öffentlich verfügbar und war endgültig für Juni 1997 angekündigt.

4.2.4 CORBA als Softwarebasis für JAVA

In der vorgestellten Anwendung führt ein mit JAVA als Applet realisierter Klient einen Request auf einem, vom IDL-Compiler für eine IDL-Schnittstellendefinition generierten Stub, aus. Dieser Stub ist mit Hilfe des JavaIDL-Compilers erstellt worden, der zur Zeit in einer frühzeitigen Testversion (alpha2) verfügbar ist. Diese JAVA IDL bildet CORBA-Objekte in JAVA-Klassen ab. Die so erstellten Stubs rufen den ORB-Kern auf, der festlegt, welcher Transportmechanismus verwendet werden soll und welche Marshallingparameter Anwendung finden.

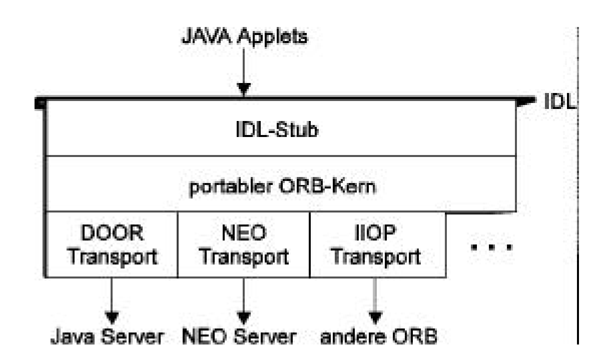


Abb. 34: Der portable ORB-Kern.

Der von SUN Microsystems vorgestellte portable ORB-Kern „Door ORB“ auf JAVA-Basis unterstützt verschiedene ORB-Protokolle. Das MarshalBuffer-Objekt verwendet Datenformate entsprechend des Ziel-ORBs, und das Repräsentationsobjekt sendet die zugehörige Anforderung unter Verwendung des ORB-Protokolls (dies ist eine vereinfachte Darstellung; die zeitgleiche Ausführung von verschiedenen Operationen durch unterschiedliche ORBs und unterschiedliche Ausnahmebehandlung muß zusätzlich berücksichtigt werden).

„Door ORB“ ist ein RPC Call/Reply-Protokoll für verteilte Objekte, das auf Basis des TCP/IP implementiert ist. Unter Verwendung von sogenannten „*Door Handles*“ werden Serverobjekte identifiziert, wobei der ORB die Vergabe von diesen Handles an Klienten protokolliert, und nur diesen Klienten den Zugriff auf assoziierte Serverobjekte erlaubt. Klienten können Handles untereinander weitergeben. Die Laufzeitumgebung des ORB verständigt dabei automatisch den Zielserver, um die Autorisation des Zugriffs durch die Klienten im Anschluß zu verifizieren. Durch Zählen der aktiven Verbindungen ist eine verteilte Garbage Collection implementiert, bei welcher der ORB-Kern den entsprechenden Server informiert [SUN96].

Um die Last auf stark frequentierten Servern zu reduzieren, wird zwischen aktiven und passiven Verbindungen unterschieden. Eine *aktive Verbindung* ist eine Verbindung, die zur Zeit für verteilte Aufrufe verwendet wird; jede dieser Verbindungen hat einen offenen TCP Stream assoziiert. Dagegen ist eine *passive Verbindung* seit einer gewissen Zeit (vorgegeben sind 60s) nicht verwendet und der zugehörige TCP Stream abgebaut worden. Dabei überwacht der „Door ORB“ weiterhin den Zustand der Verbindung, unter anderem die Zugriffsberechtigungen der Klienten. Eine *passivierte Verbindung* wird automatisch bei Bedarf reaktiviert. Dieses Verhalten kann auch durch den API-Aufruf `sunw.door.Orb.enableRecyling` erzwungen werden. Alle offenen Verbindungen werden jede halbe Stunde auf Lebendigkeit hin überprüft.

IIOP war von SUN für die JDK 1.1 angekündigt, ist aber noch nicht implementiert. Somit steht zur Zeit lediglich der Door Transportdienst zur Verfügung. Als separate Klassenbibliothek steht in der „Early Access Release“ JavaIDL 1.1 EA eine Implementierung des CORBA Naming, Lifecycle und Event Notification Service und die Unterstützung für IIOP V1.0 zur Verfügung. Das IIOP Protokoll ist, neben der Kapselung von IDL Aufrufen durch HTTP, die einzige Möglichkeit, Firewalls zu überwinden.

Zur Zeit wird JavaIDL 1.1 EA auf folgenden Plattformen mit entsprechender Software unterstützt:

- Microsoft Win32 Plattform: WinNT 3.5.1 / WinNT 4.0 / Win95
JDK 1.0.2 oder höher
Visual C++ 4.0 oder höher
- SPARC/Solaris Plattform: Solaris 2.3 oder höher
JDK 1.0.2 oder höher

In dieser Testversion wird jetzt der COSS Namensdienst unterstützt, um Objektreferenzen zu speichern und unter ihrem Namen durch „Context“ Objekte wiederzufinden. Die sehr einfache

Implementierung realisiert einen JavaIDL Name Server. Ein Applet oder eine Anwendung muß für die Verwendung dieses Dienstes den Namen und den Port des Hosts, auf dem der Server läuft, wissen. JavaIDL Namensobjekte sind nicht persistent. Der Server muß vor der ersten zugreifenden Anwendung manuell gestartet werden. Als Standardvorgabe operiert der JavaIDL Name Server auf Port 900.

Die folgende Beispielimplementierung basiert auf JavaIDL 1.0. In dieser Version stand noch kein Namensdienst zur Verfügung, so daß die URL und der Port des Serverobjektes explizit im Klienten angegeben werden müssen. Die Verwendung dieser einfachen Variante hat den positiven Nebeneffekt, daß diese ohne Modifikationen durch ein Application-Level Gateway geschleust werden kann. Der Server publiziert seine Objektreferenz direkt im ORB, die dann vom Klienten nachgefragt werden kann. Die beiden zugehörigen Funktionen sind:

```
Orb.publish(String, RemoteObjectRef)
Orb.resove(String, RemoteObjectRef)
```

Diese Einschränkung hat im Kontext von JAVA allerdings keine große Bedeutung, da JAVA-Applets ohnehin nur eine Verbindung zu ihrem originären WWW-Server aufbauen können. Auf diesem muß also auch der verwendete Namensdienst laufen, oder ein Verfahren zur Simulation eines Stellvertreters, zum Beispiel durch ein CGI Skript implementiert werden (vgl. Kapitel 2.3.5).

Zur Zeit existiert noch kein verabschiedeter Standard zur Abbildung von IDL Spezifikationen auf JAVA. JavaSoft Inc. und IONA haben bisher ihre eigenen, stark übereinstimmenden Mappings veröffentlicht. Die folgende Tabelle gibt einen Auszug der Spezifikation von JavaSoft, die voraussichtlich den Standard vorgeben werden [SUN297].

IDL-Typ	***JAVA-Typ***
boolean	boolean
char	char
octet	byte
string	java.lang.String
short	short
long	int
unsigned long	int
long long	long
unsigned long long	long

Tab. 16: Mapping der IDL-Basistypen nach JAVA.

IDL Basistypen werden auf die entsprechenden JAVA Typen abgebildet. Dieses Language Mapping ist in der „OMG-IDL JAVA-mapping submission (orbos/97-02-01)“ definiert und liegt der OMG vor.

4.2.5 Implementierung

Die exemplarisch implementierte Anwendung hat zum Ziel, Anwendern des Intra- oder Internets einen „Active View“ auf einen Datenbestand zu ermöglichen. Da innerhalb der Restriktionen dieser Arbeit nur eine 3-Tier-Realisierung möglich ist, werden die einzelnen Komponenten als CORBA-Objekte realisiert. Im Unterschied zu [ROD97] kann damit in dieser Implementierung die Komponentenkommunikation nicht durch Sockets, sondern unter Verwendung von verteilten Objekttechnologien realisiert werden.

Die exemplarische Implementierung des „Active View“ durch verteilte JAVA Objekte unterliegt besonderen Einschränkungen, die sich in drei Schritten verdeutlichen lassen.

Da der Datenzugriff auf den MS SQL Server über die lokale ODBC-Schnittstelle erfolgen soll, kann ein JAVA-Applet diesen nativen Aufruf auf dem lokalen Dateisystem, aufgrund der auf-

erlegten Sicherheitsrestriktionen, nicht direkt durchführen. Der Zugriff auf die JDBC-ODBC Brücke kann nur durch eine JAVA-Anwendung erfolgen. Dadurch ist eine 3-Tier-Struktur der Implementierung vorgegeben. Diese JAVA-Anwendung soll im weiteren als JAVA-Server bezeichnet werden. Die Kommunikation erfolgt über den „Door ORB“. Da an dieser Stelle auf die Verwendung von speziellen CORBA Diensten verzichtet werden kann, reicht dieser sehr einfache ORB aus, um die Vor- und Nachteile der Implementierung zu verdeutlichen [ROD97].

In der Weiterführung dieser Überlegung muß der JAVA-Server auf dem WWW-Server laufen da ein Applet nur eine Socket-Verbindung zu dem WWW-Server aufbauen kann, von dem es initial geladen wurde. Die Möglichkeit des Einsatzes eines Stellvertreterprogramms (z.B. als CGI-Skript realisiert), welches die Weitergabe von Socket-Aufrufen an eine weitere Maschine durchführt, ist auch hier gegeben. Da die für die Kommunikation verwendeten Ports bekannt sind, kann die Kommunikation zwischen JAVA-Klient und -Server auch durch ein Application-Level Gateway geschleust werden, bevor sie an den JAVA-Server weitergegeben wird. Dadurch ist eine minimale Sicherung des Servers möglich.

Um den letzen Schritt der Entwurfsüberlegungen durchführen zu können, muß zunächst der Begriff des *Triggers* dargestellt werden. Der MS SQL Server ab Version 6.0 implementiert diesen Mechanismus, der im Grunde nur eine spezielle Form einer gespeicherten Prozedur ist. Ein Trigger wird aktiviert, sobald die daran gebundenen Tabellen modifiziert werden. Sie werden häufig eingesetzt, um referentielle Integrität und Konsistenz zwischen logisch verknüpften Daten in verschieden Tabellen zu implementieren. Für die Implementierung des „Active View“, also einer stets aktuellen Sicht auf die präsentierten Daten, kann der Klient die Datenbank in regelmäßigen Abständen nach Datenänderungen befragen. Dieses „Polling“ führt allerdings zu unnötiger Netzlast, die besonders bei selten modifizierten Daten überflüssig und, besonders bei der Arbeit auf engbandigen Netzen, zu vermeiden ist. Eine alternative Lösung kann durch Verwendung von Triggern erfolgen, welche den Klienten auf Datenänderungen hinweisen. An dieser Stelle ist eine Form von Interprozeßkommunikation zwischen Datenbank und JAVA-Server erforderlich, die wiederum als nativer Aufruf nur von einer JAVA-Anwendung durchgeführt werden kann [ROD97].

Aus diesen drei Folgeüberlegungen ergibt sich folgendes Blockmodell für die Implementierung.

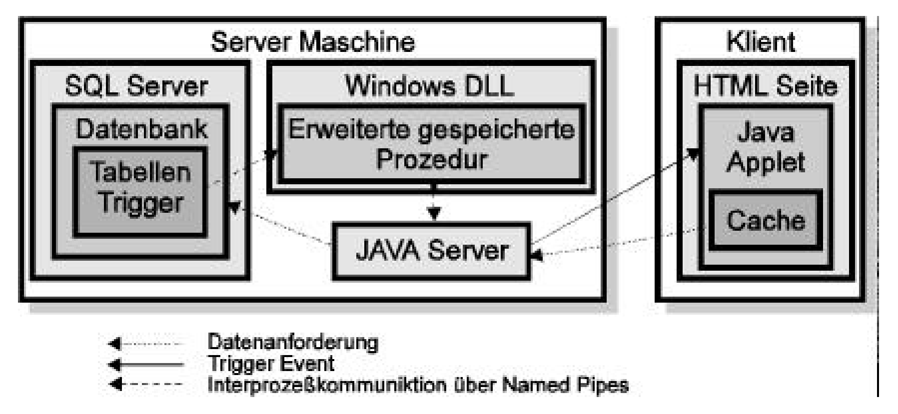


Abb. 35: Anwendungsrealisierung unter JAVA, C++ und CORBA.

Um die Implementierung im Einzelnen zu erläutern, wird die 3-Tier-Architektur von unten, schrittweise aufsteigend durchgegangen.

Auf dem MS SQL Server, der die angeforderten Datenbanken „GTZ“ hält, muß zunächst der Trigger mit der folgenden (vereinfachten) SQL-Anweisung erstellt werden

```
CREATE TRIGGER tr_triggerhelper ON GTZ FOR UPDATE AS
  DECLARE @ID int
  SELECT @ID=i.ID FROM inserted i
  EXECUTE master..triggerhelper @ID
GO
```

Der Trigger `tr_triggerhelper()` wird immer dann ausgeführt, wenn eine Modifikation an der Beispieltabelle GTZ erfolgt ist. In einem solchen Fall wird die Variable @ID deklariert, ihr die ID des modifizierten Datensatzes zugewiesen und die gespeicherte Prozedur `triggerhelper` mit dem Parameter @ID aufgerufen. Diese gespeicherte Prozedur ihrerseits wird mit der Anweisung

```
exec sp_addextendedproc 'triggerhelper', 'TriggerHelper.dll'
```

erstellt und verweist lediglich auf die C++ TriggerHelper DLL. Der Code dieser MS Windows DLL ist in der Anlage zu finden und dort dokumentiert. Beachtenswert an dieser Lösung ist, daß sie der zugrundeliegenden Tabelle keinerlei Vorgaben oktroyiert. Damit kann eine einheitliche Lösung für verschiedene Datenbestände eingesetzt werden. Da die DLL innerhalb des Prozeßraumes des MS SQL Servers läuft, muß für die Kommunikation mit dem JAVA-Server eine Interprozeßkommunikation aufgebaut werden. Unter den möglichen Vorgehensweisen wie „Shared Memory“ und Semaphoren wurde hier aus Gründen der Einfachheit und Portabilität der „Named Pipes“-Mechanismus von WinNT ausgewählt [ROD97]. Zusätzlich unterstützt der MS SQL Server bei der IPC über das „Named Pipes“-Protokoll eine sichere Kommunikation. Realisiert wird dieser IPC durch eine FIFO Datei „\\.\pipe\ssnpl“ im Systemverzeichnis des MS SQL Servers.

Die TriggerHelper DLL implementiert beide Seiten dieser Named Pipe. Die eine, die durch das Trigger Ereignis auf dem MS SQL Server mittels der gespeicherten Prozedur `triggerhelper` aufgerufen wird, und die andere, derer sich der JAVA-Server als native Methode wie folgt bedient (siehe auch [CAM97] zu nativen Methoden unter JAVA).

```
System.loadLibrary("TriggerHelper");
```

Dieser Code ist Teil der statischen NativeMethodShell Klasse, welche die DLL lädt und deren Methoden anderen JAVA-Klassen öffentlich zur Verfügung stellt. Die MS Windows DLL kann ihrerseits JAVA-Methoden aufrufen:

```
execute_java_dynamic_method(0, (HObject *)hCurrent,
                            "displayTrigger",
                            "(Ljava/lang/String;)V", hIDString);
```

Dabei ist hCurrent das Handle des JAVA Objektes, dessen Methode ausgeführt werden soll. Diese wird bei der Instantiierung der NativeMethodShell der DLL übergeben. Der vierte Parameter `(Ljava/lang/String;)V` gibt an, daß ein JAVA String Objekt übergeben und keine (Void) Parameter erhalten werden. Abschließend wird die Übergabevariable angegeben. Somit sind beide Seiten der IPC und deren unidirektionale Kommunikation implementiert. Beachtenswert ist, daß das übergebene Handle `hCurrent` des JAVA Objektes nach der Übergabe nicht gelöscht werden darf. In diesem Fall würde die JAVA „Garbage Collection“ die Referenz löschen und die MS Windows DLL würde auf einer Referenz operieren, die auf NIL zeigt [ROD97].

Um die vorliegende Implementierung nachvollziehen zu können, ist die Datei SRV.H (Open Data Services Header Datei) aus dem Microsoft Backoffice SDK oder dem Microsoft Plattform SDK für MS Windows [MSSDK97] notwendig. Sie enthält weitere Header Dateien, die im folgenden aufgeführt werden.

Header Datei	*Inhalt*
SRV.H	Alle folgenden Header Dateien. Diese Datei muß als einzige eingebunden (#include) werden.
SRVAPI.H	Definitionen von Funktionsprototypen für alle Open Data Services Funktionen.
SRVCONST.H	Open Data Services Konstantendefinitionen.
SRVDBTYP.H	DB-Library Definitionen.
SRVMISC.H	Open Data Services verschiedene Definitionen.
SRVSTRUC.H	Open Data Services Strukturdefinitionen.
SRVTOK.H	Open Data Services Token Definitionen.
SRVTYPES.H	Open Data Services Definitionen der Datentypen.

Tab. 17: Inhalte der Open Data Services Header Datei.

Diese Dateien sind Teil des MS Plattform SDK und können nicht frei weitergegeben werden; sie sind damit auch nicht im Anhang enthalten.

Die „Open Data Services (ODS)“ fungieren als Verwalter der Netzwerkverbindungen des MS SQL Server. In der Vergangenheit wurden ODS und SYBASE Open Server konzeptionell als Front-end des MS SQL Server angesehen. Es waren allerdings lediglich verschiedene Implementierungen derselben Aufgabe. Per Definition ist der MS SQL Server dagegen selbst eine ODS-Anwendung. Er verwendet die gleichen ODS Bibliotheken (OPENDS*.DLL) wie jede ODS-Anwendung. ODS übernimmt die Aufgabe der Netzwerkadministration, wartet auf neue Verbindungen, bereinigt abgebrochene Verbindungen, quittiert „*Attentions*“ (erzwungene Befehlsabbrüche) und gibt ResultSets, Nachrichten und Statusmeldungen zurück an den Klienten [MSSDK97]. MS SQL Server Klienten kommunizieren mit dem Server über das proprietäre Protokoll „Tabular Data Stream (TDS)“. TDS ist ein selbstbeschreibender Datenstrom. Dieser enthält sogenannte „Tokens“, die Namen der Spalten, Datentypen, Ereignisse (Events) und Rückgabestati beschreiben. Auf der Serverseite schreibt der MS SQL Server über den ODS direkt in TDS. Auf der Klientenseite verpacken offene Schnittstellen der DB-Library und des ODBC Paketes SQL Aufrufe in das TDS Format. Sowohl DB-Library und ODBC verwenden dafür eine Implementierung der NET Bibliothek für den Klienten [MSSDK97].

Durch dieses Vorgehen ermöglicht die TriggerHelper DLL eine Kommunikation zwischen Klienten und MS SQL Server. Um dieses aufwendige Vorgehen und die Notwendigkeit der Verwendung nativer Schnittstellen, besonders vor dem JAVA-Server (als Klienten des MS SQL Servers), zu verbergen, stellt die DLL folgende Methoden zur Kommunikation zur Verfügung:

```
int CreateServerPipe()
int DestroyServerPipe()
int ListenServerPipe()
int CreateClientPipe()
int DestroyClientPipe()
int WriteClientPipe( char *id )
```

Die Methode `WriteClientPipe()` wird von der gespeicherten Prozedur aufgerufen, und die ID des geänderten Datensatzes übergeben. Nachdem durch diesen Aufbau der MS SQL Server den JAVA-Server über die Veränderung von Datensätzen in Kenntnis setzen kann, bleiben noch zwei Aufgaben, die in den folgenden Abschnitten erläutert werden.

Zum einen muß der Datenbankzugriff mit Möglichkeiten der Modifikation durch den Klienten bereitgestellt werden. Dafür implementiert der JAVA-Server ein Datenzugriffsobjekt Remote-RecordSet, in Form einer Klasse[21]. Mit Hilfe der JDBC-API und der JDBC-ODBC Brücke stellt das so zur Verfügung gestellte Objekt „EchoImpl“ eine Repräsentation einer Tabelle dar. Diese

[21] Eine vollständige HTML Dokumentation der RemoteRecordSet-Klasse findet sich im Anhang oder unter http://www.informatik.uni-frankfurt.de/~wolter/jdbc.RemoteRecordSet.html).

Tabelle selbst ruht auf einem MS SQL 6.0 Server. Folgende Methoden wurden für diesen Test implementiert:

```
boolean r_getClientInfo(inout Info i);
boolean r_getManagementInfo(inout ManagementInfo mi);
boolean r_getCurrentRecord(inout stringSeq ls);
boolean r_updateCurrentRecord(inout stringSeq ls);
boolean r_login(in string iurl, in string iusr, in string iwd);
boolean r_createRecordSet(in string iurl, in string iusr,
                          in string iwd, in string isql);
boolean r_destroyRecordSet();
string r_getNextRecord();
boolean r_next();
unsigned long r_md_getColumnCount();
unsigned long r_md_getColumnDisplaySize(in unsigned long icol);
string r_md_getColumnName(in unsigned long icol);
```

Dabei werden die Datensätze als Sequenzen von Strings übergeben. Die Datenstruktur selbst wird dabei nicht berücksichtigt, eine entsprechende Implementierung wäre aber ohne großen Aufwand möglich. Problematisch ist die Übergabe von deutschen Umlauten über den ORB Transportdienst (im Gegensatz zu JAVA [BRU96]). Dieser kann nur ASCII Zeichen übertragen, eine entsprechende Umwandlung und Fehlerprüfung findet in der `SimpleJdbc` Klasse statt. Bei der Implementierung der `RemoteRecordSet` Klasse wurde insbesondere darauf geachtet, daß die Eigenheiten der Programmierung der JDBC-ODBC Brücke transparent werden. Weiter bieten die implementierten Methoden eine ähnliche Basisfunktionalität wie die Komponenten für den Datenzugriff unter Visual Basic. Damit ist ein direkter Vergleich in dieser Arbeit möglich, auch wenn auf die Implementierung komplexer Routinen (z.B. Offline-Cache) verzichtet wurde.

Bei diesem Vorgehen ist, neben den bekannten Vorteilen einer verteilten Implementierung, besonders ein Aspekt hervorzuheben: Im Gegensatz zu einer Inkorporation der JDBC-API im Klienten, die eine Installation einer ODBC-Schnittstelle voraussetzen würde, stellt dieser Ansatz keine Bedingungen an das Klientensystem. Es muß lokal, also auf dem System auf welchem der Browser mit dem JAVA-Applet läuft, kein ODBC-Treiber installiert sein. Eine solche Forderung würde ohnehin den großen Vorteil der Plattformunabhängigkeit zunichte machen. Die Bereitstellung der ODBC-Schnittstellendefinition über Microsoft-proprietäre Win32-APIs ist, mit den beschränkten Mitteln eines JAVA-Applets, ohnehin nicht möglich. Das eingegebene Paßwort wird bei Verwendung des „Door ORBs" in Klartext übertragen. Eine Alternative würde zum Beispiel eine Kombination der Produkte OrbixWeb und OrbixSecurity bieten. Diese kommerzielle Lösung würde die Datenkommunikation zwischen dem JAVA-Klienten und dem -Server schützen.

Zum anderen muß der Klient Informationen über Veränderungen am Datenbestand vom JAVA-Server weitergereicht bekommen. Da der „Door ORB" (wie auch die ORBIX Produktserie) keinen Ereignisdienst zur Verfügung stellt, wird hier ein Trick angewandt. Der JAVA-Klient fungiert seinerseits als CORBA Server und veröffentlicht dabei eine einzelne `Ping()`Methode. Diese Methode kann vom JAVA-Server (der jetzt die Rolle des Klienten übernimmt) aufgerufen werden. In der obigen Abbildung (Abb. 35) ist dieser Aufruf als „TriggerEvent" abgebildet. Dabei wird bewußt auf die Übertragung von zusätzlichen Informationen, wie die ID des geänderten Datensatzes oder des geänderten Datensatzes selbst, verzichtet. Bei der Operation auf engbandigen Netzen und bei hoher Dynamik der Datenbank ist die Gefahr von überhöhter Netzlast zu groß. Bei einem bloßen Ping mit der minimalen semantischen Bedeutung, daß „ein Datensatz verändert wurde", kann der Klient entscheiden, ob und wie er als Reaktion auf diese Information aktiv werden muß. Mögliche Reaktionen sind Benachrichtigung des Anwenders oder Aktualisierung der lokal zwischengespeicherten Daten mit einem oder beliebig vielen Datensätzen (an diese Stelle würde ein „Prefetching"-Algorithmus ansetzen).

Grundsätzlich ist noch zu klären, warum auf der zweiten Schicht ein JAVA-Server implementiert wird. Das vorgestellte Modell wäre erheblich einfacher ausgefallen, wenn die Funktionalität des JAVA-Servers direkt in der MS Windows DLL implementiert worden wäre. Darüber hinaus läuft

die DLL ohnehin nur auf WinNT Systemen, da die unterstützten Named Pipes zur IPC proprietär sind. Eine Implementierung zum Beispiel in C++ oder Visual Basic wäre also möglich. Folgende Gründe sprechen jedoch für die gewählte Vorgehensweise [ROD97]:

- Die Implementierung in JAVA ist schneller und einfacher.
- Der JAVA-Server ist portabel. Die MS Windows spezifischen Teile der Implementierung können dadurch reduziert werden. Auch kann bei der Implementierung von standardisierten IPCs oder von Sockets diese zweite Schicht auf einer separaten Maschine laufen und so die Verarbeitungslast von dem Datenbankserver nehmen. Hier entfällt die Bindung an das MS Windows Betriebssystem.
- Die Entscheidung für die JavaIDL führte automatisch zu einer Implementierung des Servers in JAVA, da ein IIOP Protokoll noch nicht implementiert ist.

4.2.6 Bewertung der Implementierung

Das hier vorgestellte Beispiel hat sich als sehr aufwendig in der Entwicklung herausgestellt. Neben den Unzulänglichkeiten des hier verwendeten „Door ORB“, wie sehr einfacher Namensdienst und fehlender Sicherungsdienst, sind besonders folgende Schwachpunkte aufzuzeigen:

- Durch den Einsatz von JAVA ist, auch unter Verwendung von JavaIDL, nur ein Verbindungsaufbau zum ursprünglichen WWW-Server möglich. Damit bietet diese Lösung keine „echte“ Plattform für verteilte Objekte. Der Standort des Servers ist durch das Sicherungskonzept von JAVA fest vorgegeben.

- Die Vermittlung der CORBA-Methodenaufrufe durch eine application-level Firewall ist zwar durch die von SUN vorgeschlagene CGI-Lösung prinzipiell möglich, an dieser Verbindungsstelle droht aber das Entstehen eines Flaschenhalses. Jegliche Klienten-Server Kommunikation muß über diesen, sehr langsamen Stellvertretermechanismus abgewickelt werden. Die dadurch entstehende, potentielle Sicherheitslücke in der Firewall muß analysiert werden.

- Da die ODBC-Schnittstelle als API, auch mit der darüber liegenden JDBC-ODBC Brücke, Teil des Betriebssystems ist, wird eine direkte Kommunikation zwischen JAVA-Applet und ODBC-Klienten auf der Klientenseite nicht möglich sein. Eine 3-Tier-Lösung ist unumgänglich, auch wenn auf der mittleren Schicht keine verarbeitenden Aufgaben nötig wären (wie es in diesem Beispiel der Fall ist). Abhilfe kann nur eine abgestufte Konfiguration des Sicherheitsmanagers schaffen.

- Mit die Verwendung der RemoteRecordSet-Klasse ist latent ein Datenbestand auf dem Klienten vorhanden. Durch die Implementierung eines Benachrichtigungssystems, welches den Klienten auf veränderte Daten auf dem Server hinweist, erfährt das Gesamtsystem eine Verzögerung. Während der Zeit, in dem der Trigger vom Server zum Klienten läuft, werden die Daten auf dem Klienten veraltet dargestellt. Dieses Problem kommt vor allem bei sehr dynamische Datenbeständen und langsamen Netzwerkverbindungen zum Tragen.
  Dieser mangelnde Funktionsumfang ist nicht zuletzt in den beschränkten Möglichkeiten der JDBC-ODBC Brücke verankert. Einschränkungen, wie die vorgegebene Verarbeitungsrichtung der Daten „von links nach rechts und von oben nach unten“ und die Unfähigkeit, Datensätze wiederholt zu lesen, sind in einem produktiven Umfeld nicht vertretbar. An dieser Stelle muß die zukünftige Entwicklung ansetzten, um das vorgestellte Modell für den Einsatz in der Praxis zu modifizieren.

Neben diesen Schwächen des Modells selbst, ist die Implementierung an verschiedenen Stellen noch zu verbessern. Die vorgestellte Implementierung der RemoteRecordSet-Klasse bietet nur rudimentäre Methoden für die Administration und Modifikation von Daten. Es werden nur Zeichenketten unterstützt. Eine Unterstützung von Transaktionen ist zwar hart (durch Exceptions) kodiert, gespeicherte Prozeduren und Cursors werden allerdings nicht behandelt.

Insgesamt muß diese Klasse noch erheblich erweitert werden, um produktiv eingesetzt werden zu können.

Weiter muß untersucht werden, inwieweit die Interprozeßkommunikation, die durch die Trigger-Helper DLL aufgebaut wird, einen Flaschenhals in dem Modell bildet. Über sie werden alle Trigger-Benachrichtigungen an alle Klienten geleitet. In diesem Zusammenhang ist auch die Leistungsfähigkeit des JAVA-Servers auf der zweiten Schicht im Lastfall zu analysieren.

4.3 Visual Basic und ActiveX

4.3.1 ActiveX im Intranet und Internet

Unabhängig von dem Begriff von ActiveX als Technologie, der nach Microsoft die Interoperabilität zwischen Komponenten unter Verwendung von COM ermöglicht, ist der Begriff von ActiveX-Komponenten (Controls) zu definieren.

> *„Eine* ***ActiveX-Komponente*** *ist eine Software, die eine allgemeine Aufgabe auf standardisierte Art erfüllt* [CHAP96].*“*

Genauer definiert eine ActiveX-Komponente eine standardisierten Menge von Interfaces einer COM-Komponente. Da ActiveX-Komponenten im allgemeinen keine COM-Komponentenserver sind, beinhaltet die ActiveX Spezifikation zusätzliche Regeln für die Erstellung von *Komponentencontainern*. Dies sind Klienten, die ActiveX-Komponenten verwenden können (z.B. MS InternetExplorer ab V3.0).

Die Frage nach der tatsächlichen Bedeutung von ActiveX-Komponenten läßt sich besser in deren zeitlichen Entwicklung erkennen. Die Idee von wiederverwendbaren, binären Komponenten begann bei Microsoft mit der Implementierung von „Visual Basic Extensions (VBX)“. Diese frühen Komponenten boten sehr viele Merkmale von ActiveX-Komponenten, waren aber nur für den Einsatz innerhalb der Entwicklungsumgebung von MS Visual Basic 3.0 konzipiert. VBX unterliegen allerdings Einschränkungen, nicht zuletzt dem des beschränkten Einsatzfeldes. Gerade dieses Problem wurde bei den sogenannten „*OLE Controls* (OCX)“ aufgegriffen. Dieser Standard verlangte von einer COM-Komponente die Implementierung einer Vielzahl von Interfaces (z.B. Darstellung einer eigenen GUI, Ereignissteuerung für den Komponentencontainer etc.). Diese Vorgaben führten zur Überladung der meisten Komponenten, zur Vergrößerung des Binärcodes und damit zur Nichteignung für den Einsatz im Internet.

OLE Komponenten werden jetzt als ActiveX-Komponenten bezeichnet. Diese Neubenennung geht mit der Reduzierung der Anforderungen an eine solche Komponente einher. Damit ist

> *Eine* ***ActiveX-Komponente*** *eine COM-Komponente, die das IUnknown Interface unterstützt und sich selbst registrieren kann* [CHAP96].

Durch die Fähigkeit der eigenen Registrierung in der Registry des Betriebssystems, kann die Komponente ohne separaten Installationsvorgang vom Klienten verwendet werden. Eine begriffliche Trennung zur COM-Komponente wird in der Literatur weiterhin gepflegt [CHAP96], soll auch hier zum besseren Verständnis beibehalten werden, ist aber aus Sicht der Technologie in keiner Weise sinnvoll.

Hervorzuheben bleibt noch, daß ActiveX-Komponenten sich in der Registry unter verschiedenen Kategorien eintragen (können), um ihre Mächtigkeit anzugeben. Anders als die üblichen mnemonischen String-Einträge werden hier GUID verwendet, die als „Category Identifier (CATID)“ bezeichnet werden. Im Unterschied zur CLSID definiert also eine CATID die Fähigkeit einer Komponente, eine bestimmte Anforderung (nicht Methode) erfüllen zu können [CHAP96].

Für den Endanwender bilden Komponentencontainer und ActiveX-Komponente eine Einheit, die sich als homogene Anwendung darstellt. Dies ist besonders für die Verwendung im Internet interessant, wo ein ActiveX-fähiger Browser als Komponentencontainer eingesetzt wird.

Eine ActiveX-Komponente kann durch Verwendung des OBJECT Tags in HTML geladen und verwendet werden.

```
<HTML>
  <TITLE>CONTROL EXAMPLE</TITLE>
  <BODY>
    <OBJECT CLASSID="clsid:B16553C0-06DB-101B-85B2-0000C009BE81"
     CODEBASE="http://www.someserver.com/ExampleComponentID.oxc"
     ID=ExampleComponentID>
  </BODY>
</HTML>
```

Nach dem Laden der HTML-Seite wird diese vom Browser interpretiert. Das CLASSID Attribut wird ausgelesen und in der Registry nachgeschlagen. Ist die angeforderte Komponente auf dem lokalen System vorhanden, kann direkt `CoCreateInstance()` mit der CLSID als Parameter aufgerufen werden. Falls nicht, wird die Komponente automatisch, von der unter CODEBASE angegebenen URL, auf das lokale System geladen. Dieser Vorgang wird durch den *Internet Komponenten Download Dienst* durchgeführt, der drei Verpackungsschemata für Binärcode vorsieht:

- Ein ausführbares Programm, das eine einzelne Datei mit der Endung OCX, DLL oder EXE enthält.
- Ein sogenanntes Kabinett von Dateien mit der Endung CAB. Dies ist eine Sammlung von (mehreren) ausführbaren und nicht ausführbaren Dateien und einer NIF Datei, welche Installationsvorschriften für die anderen Dateien beinhaltet. Dieses Schema findet sehr häufig Anwendung, da viele Komponenten DLLs benötigen, die zur der modularen Menge von APIs des Betriebssystems gehören, die nicht zwingend erforderlich sind.
- Eine NIF Datei mit einer Referenz auf eine Datei, die selbst wiederum geladen werden soll.

Dieser Vorgang ist gänzlich transparent, sowohl für den Anwender als auch dem Entwickler. Lediglich der Aufruf von `CoGetClassObjectFromURL()` stößt den gesamten Mechanismus an.

ActiveX verbindet dabei Skriptsprachen und Komponentenmodelle wie OLE und JAVA. MS Visual Basic wurde als erste Containersprache für ActiveX-Komponenten angeboten. MS C++ und MS Visual JAVA folgten. Der MS InternetExplorer V3.0 war der erste Browser, der dann diese ActiveX-Komponenten unterstützte.

Ein weitverbreiteter Kritikpunkt ist die mangelnde Offenheit des ActiveX- und damit des COM-Standards. Auf Initiative von Microsoft wurde die „ActiveX Group“ (http://www.activex.org) als Vereinigung von Herstellern wie DEC, HP, SAP, SNI und der Software AG im Rahmen der Open Group gegründet[22]. Wie ernst es Microsoft mit der Öffnung seines Standard meint bleibt abzuwarten. „In den ActiveX Strategiepapieren heißt es deutlich, daß das ActiveX-Konsortium sowohl von der Finanzierung als auch von der Verwaltung und dem Betrieb her, von Microsoft abhängig sein wird [CT1209].“ Spätestens seit der Publikation des DCOM Protokolls als Internet Draft [DRAFT961] muß auch ActiveX zur Zeit als offener Standard klassifiziert werden, auch wenn er große Teile der Win32-API verwendet, dessen Spezifikation nur teilweise veröffentlicht ist.

[22] Eine vollständige Liste der Mitglieder findet man unter http://www.activex.org/people/index.htm.

4.3.2 ActiveX Sicherheitskonzept

Gerade das Laden von Binärcode über ein potentiell unsicheres Netzwerk wie das Internet bringt die Frage nach der Sicherheit solcher Komponenten auf. Während JAVA den Ansatz der inhärenten Sicherheit der Sprache selbst durch das Sandbox Verfahren verfolgt, geht ActiveX den Weg über Vertrauen. Hier ist nicht das „blinde" Vertrauen gemeint, sondern das, durch das vorgestellte Sicherheitsmodell definierte Vertrauen (vgl. Kapitel 2.4.2.4), als Grundlage von Sicherheit und Akzeptanz.

Das Erlangen von Vertrauen ist das Ziel des *Windows Trust Verification* Dienstes. Mittels einer einzigen Methode `WinVeryTrust()` kann ein Anwender durch diesen Dienst einen oder mehrere *Trust Provider* befragen; ein Trust Provider kann die Antwort auf die Frage geben, ob einer Komponente unter vorgegebenen Kriterien vertraut werden kann. Genauer wird Auskunft darüber gegeben, wer diese Komponente erstellt hat und ob diese Person oder Institution vertrauenswürdig ist. Zusätzlich wird sichergestellt, daß die Komponente nach der Erstellung nicht verändert worden ist. Eine Komponente, die durch einen Komponentencontainer (hier einen Browser) geladen wird, kann digital signiert werden. Diese digitale Signatur ist ein Byte String der verwendet werden kann, um die Herkunft der Komponente zu bestimmen. Der „Windows Software Publishing Trust Provider", also die Instanz, welche die Signaturen verwaltet, verwendet dafür PKCS #7 und X.509 V3 Zertifikate [BRU97].

Der zugrundeliegende Vorgang verläuft wie folgt: Das System extrahiert die Signatur, stellt die zertifizierende Autorität, die das Zertifikat des Herstellers ausgestellt hat, fest, und erfragt den öffentlichen Schlüssel des Herstellers. Der öffentliche Schlüssel wird für die Entschlüsselung des verschlüsselten Hash verwandt. Dieser wird mit der Ausgabe desselben Einweg-Hash Algorithmus verglichen, welcher auf dem Code ausgeführt wird. Wenn der Code nach der Signatur nicht verändert wurde, sind der alte und neue Schlüssel identisch. Falls die Schlüssel nicht übereinstimmen, erhält der Anwender einen Warnhinweis und kann dann entscheiden, ob der Code ausgeführt werden soll. Wenn der Anwender sich für die Ausführung entscheidet, treten keine weiteren dynamischen Überwachungen in Kraft. Ein Versehen des Codes mit einer Spezifikation des Ressourcenbedarfs durch den Hersteller ist nicht vorgesehen [CHAP96].

Kürzliche Pressemeldungen haben das Produkt „Authenticode" der Firma Microsoft mit ActiveX Sicherheit gleichgesetzt. Dies ist inkorrekt. Authenticode arbeitet mit JAVA-Applets, Browser Plug-Ins und ActiveX-Komponenten, um für diese Softwarekomponenten eine Herstelleridentifikation und damit auch Herstellerrechenschaft zu ermöglichen. Authenticode ist eine Implementierung des Vorschlages für „Signatur von Quellcode" der Firma Microsoft an das „World Wide Web Consortium (W3C)".

Authenticode verwendet ausschließlich dokumentierte Win32-APIs. Wenn eine Komponente durch einen Browser geladen wird, erhält der Anwender einen Nachweis über die Identität des Autors und eine URL, die auf weitere Informationen verweist. Durch Authenticode können Endanwender also den Autoren Rückmeldungen geben und bei Bedarf zur Rechenschaft ziehen.

Aufregung in der Öffentlichkeit verursachte am 17. Juni 1996 Fred McLain mit seiner „Exploder Komponente" (http://www.halcyon.com/mclain/ActiveX/). Wenn diese ActiveX-Komponente über das Internet auf einem Personalcomputer mit „Power Conservation BIOS" geladen wird, beendet es Win95 und schaltet den Personalcomputer ab. Dieses Verhalten wurde als Beweis für die Unzulänglichkeit der COM-Sicherungsmechanismen und von Authenticode bewertet. Diese Schlußfolgerung ist aber nicht korrekt. Am 23. August hat McLain seine Komponente mit seiner individuellen „Software Publisher Digital ID" von dem Trust Provider VeriSign signiert und anschließend veröffentlicht [MS964]. Dadurch hat er sich als Herausgeber identifiziert und ist für die Resultate der Ausführung seiner Komponente verantwortlich.

4.3.3 Visual Basic als Softwarebasis für ActiveX

Im Unterschied zu JAVA als objektorientierte Programmiersprache, ist COM ein Objektmodell. In der Diskussion über verteilte Objekte ist ein Vergleich zwischen dem durch JAVA implementierten Modell und COM, nicht aber zwischen JAVA und der Programmiersprache, die das COM implementiert, zu ziehen. Trotzdem wird im folgenden kurz, für das allgemeine Verständnis und nicht zuletzt auch wegen dem allgemein schlechten Ruf in der Öffentlichkeit, auf die Eigenschaften von Visual Basic eingegangen.

Zunächst muß erläutert werden, warum eine Programmiersprache wie MS Visual Basic in einer Diskussion um verteilte Objekte aufgeführt werden kann. Eine objektorientierte Programmiersprache unterstützt Kapselung, Abstraktion, Polymorphismus und Vererbung. Man kann MS Visual Basic aber als objektzentriert bezeichnen [SPE96], da es sowohl den Begriff des Objektes als auch den der Klassen unterstützt. Allerdings wird Vererbung nicht von MS Visual Basic unterstützt [ROM97]. Der entscheidende Grund für die Verwendung von MS Visual Basic ist aber seine weite Verbreitung. „IS Abteilungen steigen zunehmend auf Visual Basic um und machen es damit zu der am häufigsten verwendeten Programmiersprache seit COBOL [KIR95]“.

Die folgende Tabelle zeigt die Anzahl der installierten Entwicklungsumgebungen im Vergleich.

Sprache	***Installierte Basis***
COBOL	2,5 Millionen
Visual Basic	2,0 Millionen
C++	1,2 Millionen
Smalltalk	0,1 Millionen

Tab. 18: Marktanteile verschiedener Programmiersprachen [KIR95].

Weiter hat Microsoft, durch die Veröffentlichung von Visual Basic „Control Creation Edition (CCE)“, diese Programmiersprache zur firmenstrategischen Plattform für ActiveX erklärt [MS951].

4.3.4 Datenbankanbindung in Visual Basic

4.3.4.1 Zusammenfassung

Visual Basic stellt eine Vielzahl von Mechanismen zur Datenanbindung für den Entwickler zur Verfügung. Für den direkten Zugriff auf ODBC-Datenquellen kommen „Data Access Object (DAO)“ und „Remote Data Object (RDO)“ in Frage. Der direkte und komplizierte Zugriff auf die ODBC-API kann dadurch umgangen werden. Die folgende Abbildung verdeutlicht diesen Zusammenhang. Auf die Implementierung einer VBSQL Schnittstelle wird verzichtet, da diese eine Abhängigkeit zum MS SQL Server schaffen würde. Ein Einsatz von anderen ODBC Datenbanken wäre so nicht möglich (Informationen zu diesem Thema finden sich unter http://www.microsoft.com/vbasic und http://www.microsoft.com/odbc).

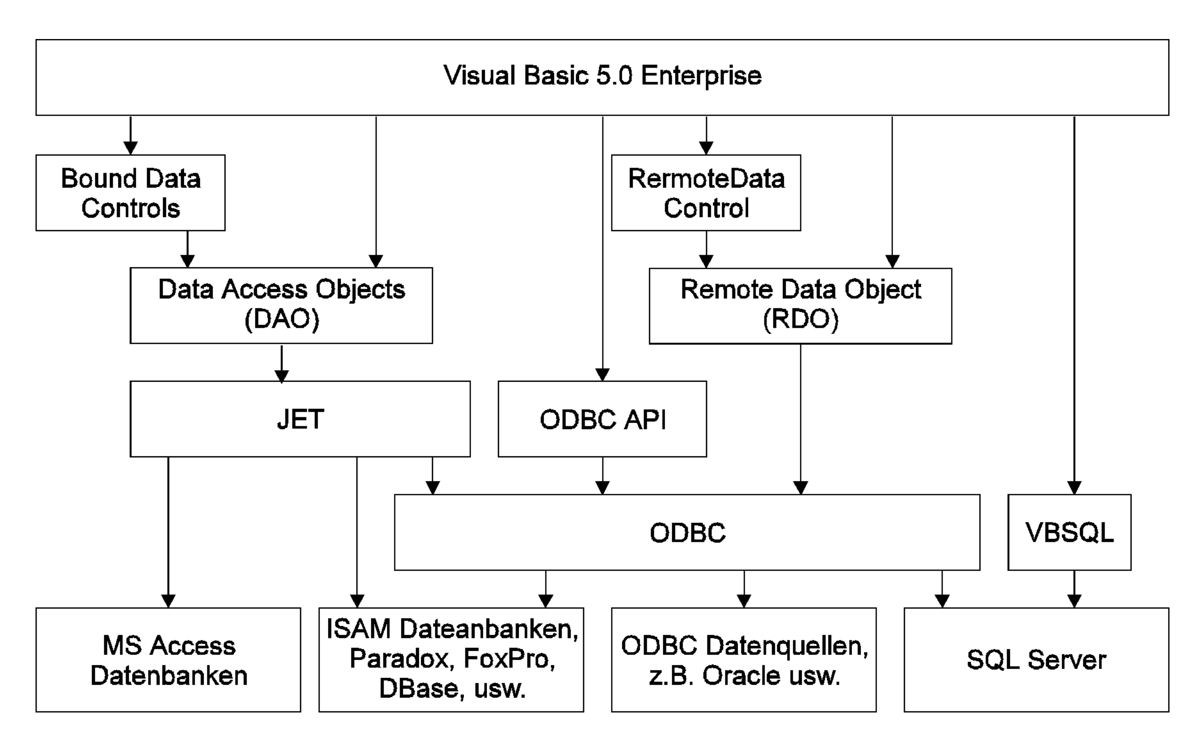


Abb. 36: Optionen des VB5.0 Datenzugriffmodells.

Die Verwendung der ODBC-API hat einen erheblichen Aufwand für die Implementierung zur Folge, der nicht mit einem direkten Gewinn an Performanz verbunden ist. Aus diesem Grund ist eine solche Implementierung für Standardaufgaben nicht sinnvoll. Die beiden verbleibenden Alternativen werden im folgenden vorgestellt.

4.3.4.2 Das Data Access Object

Das DAO stellt ein einfaches Objektmodell für die von der Datenquelle unabhängigen Datenanbindung zur Verfügung. Diese muß lediglich eine DAO Schnittstelle unterstützen, was bei ODBC-Datenquellen der Fall ist. Einige DAO-Objekte, -Methoden und -Eigenschaften sind explizit für die Implementierung und Unterstützung der „Indexed Sequential Access Method (ISAM)“-Struktur von Jet und installierbaren ISAM-Datenbanken konzipiert (ISAM ist ein Schema, das die Zeit für das Finden von Datensätzen in großen Datenbanken, durch Verwendung eines eindeutigen Schlüssels, reduzieren soll [RIC90]). Der Zugriff über DAO auf ODBC-Datenquellen ist damit möglich, aber nicht performant, weil auch bei dem direkten Zugriff auf ODBC-Datenquellen die (veraltete und langsame) Jet-Bibliothek Verwendung findet [MS971] (siehe dazu auch Abb. 36).

DAO ist eine Sammlung von Methoden, die den Rahmen für Code zur Datenanbindung und Datenmanipulation bereitstellen. Die entsprechenden Objekte und Collections haben Eigenschaften, welche die Charakteristika von Datenbankkomponenten und Methoden zur deren Manipulation beschreiben. Zusammen bauen diese Objekte und Collections ein hierarchisches Modell der zugrundeliegenden Datenbankstruktur auf, das sehr pragmatisch aus der Entwicklungsumgebung heraus kontrolliert werden kann. Eine Auflistung der Methoden ist im MS DAO SDK 3.5 als Teil des MS Developer Network nachzulesen [MSSDK97].

Das DAO Programmiermodell bietet folgende Funktionsmerkmale:

- Ein universelles Programmiermodell, mit dem auf jede ODBC Datenbank zugegriffen werden kann - unabhängig vom Grad der ODBC-Konformität. Während RDO den Einsatz von Ebene-2 ODBC-Treibern (oder entsprechend erweiterten Ebene-1 Treiber) erfordert, besteht eine solche Voraussetzung für DAO/Jet nicht.
- Die Möglichkeit zum Abrufen und Verknüpfen von heterogenen Daten aus ODBC oder ISAM-Quellen.

- Die Fähigkeit zum Erzeugen von aktualisierbaren Cursorn aus komplexen, als Verknüpfungsprodukt erzeugten Ergebnismengen.
- Ein objektorientiertes Programmiermodell, das keine direkten API-Funktionsaufrufe benötigt.
- Einen Mechanismus zur Weiterleitung (Pass-Through), der die Umgehung der (langsamen) Jet Abfrageverarbeitung erlaubt.
- Implementierung von Cursorn des Typs „Schlüsselgruppe“ (Dynaset), „Statisch“ (Snapshot) und „Vorwärts“ (Read Ahead). Letztere bieten besonders bei langsamen Netzwerken die Möglichkeit der Steigerung der Performanz.
- Leistungsfähige Verwaltung der Ergebnismengen.
- Universelle Fehlerverwaltung.

Das DAO selbst erhält seine Funktionalität durch das DAO COM-Interface. DAO vereinfacht die Verwaltung dieser COM-Interfaces, indem es sie in eine dbDAO Klasse wie zum Beispiel CdbRecordset, CdbWorkspace, usw. integriert. Diese Vorgehensweise verbirgt das Interface, und dessen aufwendige Implementierung, für den Entwickler. Diese Transparenz ist aber zum Beispiel bei der Verwendung des DAO in einer „Multi-Threaded apartment-model“ Anwendung nicht wünschenswert, da hier der direkte Zugriff auf das Interface über die einzelnen Appartements (Threads) hinaus erforderlich ist. Normalerweise würde die Anforderung eines Interfaces durch den Aufruf von `CoCreateInstance()` erfolgen. DAO akquiriert aber bereits dieses Interfaces für eigene Operationen und bietet es wiederum durch die `GetInterface()` Methode und den LPUNKNOWN Operator an.

4.3.4.3 Das Remote Data Object

Das RDO ist eine MS API, welche Datenbankoperationen auf entfernten Datenbanken für den Entwickler transparent gestaltet. RDO legt eine dünne Codeschicht über das ODBC-API und den ODBC-Treibermanager, der die Verbindungen einrichtet, Ergebnismengen und Cursor erstellt und komplexe Prozeduren ausführt. Dabei werden die Ressourcen der Arbeitsstation nur minimal belastet. Mit RDO und dem Remote Daten Steuerelement können Anwendungen auf ODBC-Datenquellen zugreifen, ohne eine lokale Abfrageverarbeitung zu verwenden. Das bedeutet ein deutlich besseres Leistungsverhalten und mehr Flexibilität beim Zugriff auf entfernte Datenbankmodule [MS961]. Vergleichbare Leistungswerte werden nur durch Verwendung der ODBC-API direkt erreicht.

Im Prinzip werden RDO auf ähnliche Weise wie die Datenzugriffsobjekte (DAO) des MS Jet Datenbankmoduls verwendet und das Remote Daten Steuerelement (RemoteData) ist mit dem Daten Steuerelement (Data) vergleichbar. Interessant ist die Fähigkeit einer Zwischenspeicherung von Daten auf dem Klienten und die Möglichkeit des „Prefetching“, also dem vorzeitigen Laden von Daten, bevor diese tatsächlich angefordert werden. Mit geringen Änderungen können vorhandene Anwendungen, die DAO und das Daten Steuerelement verwenden, für die Verwendung von RDO und RemoteData umgewandelt werden. Es gibt einige Unterschiede insofern, als RDO für den Gebrauch in streng relationalen Datenbanken implementiert und ausgelegt ist. RDO unterstützt selbst keine Abfrageverarbeitung; für die Bearbeitung aller Abfragen und die Erzeugung der Ergebnismengen ist die Datenquelle zuständig. Die Datenobjekte selbst werden aus den Ergebnismengen und Cursorn erstellt, die vom ODBC-Treiber zurückgegeben werden.

Die Umwandlung einer vorhandenen DAO/Jet-Anwendung nach RDO kann sich aufgrund der ODBCDirect Schnittstelle erübrigen, weil diese DAO-Referenzen über RDO statt Jet leitet. Wenn die Anwendung jedoch keine DAO ISAM Objekte und Methoden (wie das Recordset Objekt vom Typ „Tabelle“ und die Seek-Methode) oder andere ISAM Programmiermethoden verwendet, sollte eine Umwandlung in Richtung ODBCDirect mit nur wenigen Änderungen möglich sein - weniger noch als bei einer Umwandlung nach RDO.

Eigenschaften des RDO:

- RDO ist ein universelles Programmierungsmodell mit dem auf beliebige ODBC-Datenquellen zugegriffen werden kann.
- Mit Hilfe von RDO können einfache Ergebnismengen ohne Cursor oder komplexe Cursor erstellt werden.
- Weiter können auch Abfragen ausgeführt werden, die eine beliebige Anzahl von Ergebnismengen zurückgeben oder gespeicherte Prozeduren ausführen, die Ergebnismengen mit oder ohne Ausgabeparameter und Rückgabewerte zurückgeben werden.
- RDO erlaubt synchrone oder asynchrone Operationen, so daß eine Anwendung nicht blockiert sein muß, wenn eine längere Abfrage durchgeführt wird.
- RDO ist ein objektorientiertes Modell, das keine Verwendung von APIs erzwingt
- Es bietet universelle Fehlerbehandlung.
- RDO erlaubt dynamische Sperrung von Tabellen, Datensätzen und Feldern.
- RDO erlaubt die Konfiguration eines lokalen Zwischenspeichers. Dabei kann die Größe dieses Speichers, der bei Bedarf automatisch aktualisiert werden kann, dynamisch konfiguriert werden. Je nach Anforderung durch den Klienten werden Daten automatisch „nachgeladen“. Die Größe des Zwischenspeichers könnte zum Beispiel durch die Netzlast variiert werden. Ein „Prefetching“ ist ebenfalls konfigurierbar.

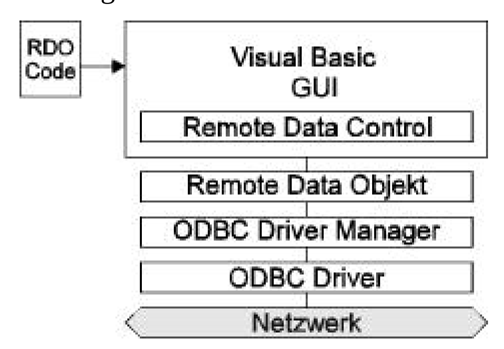


Abb. 37: Entfernter Datenzugriff über RDO.

Durch die Umgehung der lokalen Verarbeitung von Abfragen sind kaum Modifikationen erforderlich, um einen Datenbankklienten auf einer entfernten Maschine zu implementieren (ODBC-Schnittstellendefinition muß in die Registry eingetragen werden - dieser Vorgang ist aber gegebenenfalls zu automatisieren). Dadurch eignet sich dieser Ansatz besonders für den Einsatz im Internet (innerhalb einer ActiveX-Komponente). Die folgende Funktion dient der ODBC-Treiber-Initialisierung zur Laufzeit [MSKB972]. Er ist für die ODBC V 3.0 getestet[23]. Das in dieser Arbeit implementierte Beispiel basiert auf dem ODBC-Treiber V3.0.32.1. Die ODBCINST.DLL ist ein Bestandteil des Treibers.

```
Declare Function SQLConfigDatasource Lib "odbcinst.dll" _
                   (ByVal Hwnd%, ByVal fRequest%, _
                    ByVal lpszDriver$, _
                    ByVal lpszAttributes$) As Integer
Global Const SQL_Add_DataSource = 1

Function New_DataSource (DataSourceType$, DataSourceName$, _
                          ServerName$, DatabaseName$) As Variant
  Dim DName, DSNamen, Server, DDatabase As String

  DName = DataSourceType & Chr$(0)
  DSName = "DSN=" & DataSourceName & Chr$(0)
  Server = "Server=" & ServerName & Chr$(0)
  DDatabase = "Database=" & DatabaseName & Chr$(0)
```

[23] ODBC Treiber für verschiedene BS finden sich auf http://www.microsoft.com/data/odbc/download.htm.

```
        New_DataSource = SQLConfigDatasource(0&, SQL_Add_DataSource, _
                                             DName, _
                                             DSName & Server & DDatabase)
    End Function
```

Die Funktion `SQLConfigDataSource()` wird hier verwendet, um eine Datenquelle dynamisch hinzuzufügen, zu bearbeiten oder zu löschen. Diese Funktion verwendet Schlüsselwörter, mit denen Verbindungsoptionen festlegen werden, die beim Verbinden mittels der Systemsteuerung, über das Dialogfeld ODBC-Setup (vgl. Abb. 38), durch den Anwender auch interaktiv eingestellt werden können.

- DName: Name des ODBC-Treibers
- DSName: Name der Datenquelle selbst
- Server: Name des Servers (hier: auf dem MS SQL 6.0 läuft)
- DDatabase: Bezeichner der Datenbank
- Bei Bedarf können noch Anwendername (USR) und -paßwort (PWD) übergeben werden.

Die dynamische Erstellung hat den wünschenswerten Nebeneffekt, daß nach Terminierung der Anwendung die ODBC Konfiguration gelöscht, und damit dem einfachen Zugriff durch Dritte entzogen wird. Bei diesem Systemmodell ist es unerheblich, ob WWW-Server und MS SQL Server auf separaten Maschinen oder auf einem einzelnen Server laufen. Das RDO macht diesen Unterschied für den Entwickler absolut transparent.

Remote Data Objekte und -Auflistungen stellen einen Rahmen für Code zur Verfügung, um Komponenten eines Remote ODBC Datenbanksystems zu erstellen und zu verändern. Objekte und Auflistungen haben Eigenschaften, welche die Merkmale der Komponenten von Datenbanken beschreiben und Methoden, mit denen diese Komponenten verändert werden können. Mit dem aus hierarchischen Strukturen bestehenden Rahmen erstellen sie Beziehungen zwischen Objekten und Auflistungen, und diese Beziehungen repräsentieren die logische Struktur des verwendeten Datenbanksystems. Das rdoEngine Objekt repräsentiert die entfernte Datenquelle und wird automatisch erstellt, wenn ein beliebiger Verweis auf RDO oder das Remote Daten Steuerelement erstellt wird. Wenn RDO beim ersten Zugriff initialisiert wird, erstellt RDO automatisch eine Instanz von rdoEngine und der Standardumgebung `rdoEnvironments(0)` [MS962].

Eine exemplarische Realisierung einer einfachen Oberfläche führt den Anwender, nach Bestätigung der Sicherheitsabfrage des MS InternetExplorers, zur Eingabemaske des MS SQL Server Logins. Paßwörter werden in diesem System zentral vergeben. Bemerkenswert ist, daß die Übermittlung der kritischen Anmeldeinformation nicht über das (unsichere) TCP/IP sondern über „sichere“ RPC verschlüsselt durchgeführt wird.

Abb. 38: ODBC Anmeldungsdialog.

Nach der Authentifizierung durch den Datenbankserver wird die ActiveX-Komponente (OCX Datei im Beispiel 13KB) auf das System des Klienten bei Bedarf kopiert beziehungsweise aktualisiert und ausgeführt.

RDO wurden entworfen, um auf externe Datenbanken anstatt auf lokale ISAM Datenbanken zugreifen zu können. Datenzugriffsobjekte (DAO) stellen die beste Wahl für die Arbeit mit ISAM Daten dar, da sie mit der gleichen Semantik wie ISAM Daten erstellt wurden (Suchmethoden, öffnen von Tabellen anstatt ausführen von Abfragen). Die Verwendung von RDO bei der Kommunikation mit ISAM Daten bringt dagegen keine Vorteile, da der ODBC-Treiber das Jet Modul dennoch lädt.

Die Unterschiede, die bei verteilten Anwendungen für das RDO sprechen, sind in der folgenden Tabelle zusammengefaßt [MS971]:

Leistungsmerkmal	***DAO***	***RDO***
Verbindungsaufbau (erste Verbindung)	5-8s (Schnellverbindung 3-6s)	3-6s (Schnellverbindung 2-5s)
Asynchrone offene Verbindungen	Nein	Ja
Asynchrone Abfragen	Nein	Ja
Stored Procedure Result Codes	Nein	Ja
Benachrichtigung per Ereignis	Nein	Ja
Unterstützung von Cursorn (unter MS SQL 6.0)	Nein	Ja
Wiederholung nach Zeitüberschreitung	Nein	Ja
Dynamischer / Statischer Zwischenspeicher	Nein / Nein	Ja / Nein

Tab. 19: Unterschiede zwischen RDO und DAO.

Gerade diese Leistungsmerkmale sind bei dem Zugriff auf entfernte Datenbanken unverzichtbar. Die Vorteile, die das RDO dem klassischen DAO voraus hat, wiegen den möglichen Aufwand für die Neuimplementierung von Anwendungen auf [MSDBLIB].

4.3.5 Implementierung

Die „Enterprise“ Version von Visual Basic 5.0 ermöglicht die Verwendung des COM auf Komponentenebene. Somit wird die zeitaufwendige Implementierung auf API Ebene vermieden, ohne dabei Funktionalität zu verlieren [MS951]. Grundsätzlich ist zunächst klarzustellen, daß eine Realisierung eines „Active View“ unter ActiveX nicht den starken Restriktionen einer JAVA VM unterliegt. Somit ist es möglich, daß eine ActiveX-Komponente direkt mit dem MS SQL Server kommuniziert. Auch die Konfiguration der Datenquelle für den ODBC Zugriff kann, wie beschrieben, dynamisch vorgenommen werden, sofern die ODBC-Treiber selbst auf dem Klientensystem installiert sind (davon kann zumindest bei MS Windows Betriebssystemen ausgegangen werden). Darüber hinaus besteht durch Verwendung des RDO die Möglichkeit, eine direkte und sichere Verbindung zu MS SQL Server über Sockets (Standardwert des Ports 1433 ist die offizielle Socketnummer des „Internet Assigned Number Authority (IANA)“ für MS SQL Server) oder das sogenannte MultiProtokoll [MSSQL96] zu realisieren. Letzteres findet in der vorgestellten 3-Tier-Realisierung Verwendung, da das „Multi Protocol Net Library (MPNL)“ die Vorteile des abgesetzten RPC von WinNT nutzt. Im Gegensatz zu anderen Netzbibliotheken muß die MPNL lediglich als Option bei der Konfiguration des MS SQL Servers gewählt, und nicht weiter konfiguriert werden. Sie erfüllt folgende Funktionen [MSSQL96]:

- Sie ermöglicht die Kommunikation über die meisten von WinNT unterstützen IPC Mechanismen.
- Sie ermöglicht die Verwendungen der integrierten Sicherheitsfunktionen, die RPC unterstützt.

- Sie unterstützt Verschlüsselung für Benutzerkennwörter und Daten.
- Im Gegensatz zu Named Pipes, NWLink, Banyan Vines und TCP/IP Sockets unterliegt sie keinen Verbindungsbeschränkungen.
- Sie bietet eine Leistung, die für die meisten Anwendungen mit systemeigenen IPC Netzbibliotheken vergleichbar ist.

Als 3-Tier-Beispielimplementierung fallen der mittleren Schicht lediglich zwei Aufgaben zu. Zum einen veröffentlicht der Server die beiden Methoden

```
Public Function CreateRDO(sSQL As String) As RDO.rdoResultset
Public Function CreateDAO(sSQL As String) As DAO.Recordset
```

Diese dienen lediglich der Initialisierung der Datenkomponenten, die wiederum durch den Server veröffentlicht werden. Mit dem folgendem Codesegment instantiiert der Klient das Serverobjekt, welches unter dem Namen DCOMDataServer mit der Klassenbezeichnung DataServer in der Registry eingetragen ist, und initialisiert anschließend das Datenbankobjekt.

```
Dim mobj As Object
Set mobj = CreateObject("DCOMDataServer.DataServer")

Dim mrdo As RDO.rdoResultset
Set mrdo = mobj.CreateRDO("select * from gtz")

Dim mdao As DAO.Recordset
Set mrdo = mobj.CreateDAO("select * from gtz")
```

Für diesen Versuch wurde auf der Serverseite sowohl das DAO als auch das RDO implementiert. Aus den bereits erläuterten Gründen ist der Einsatz des RDO allerdings vorzuziehen. Hier soll nur dargestellt werden, daß sich die Datenbankobjekte für den Klienten identisch verhalten. Nur der vorbereitende Code im Server unterscheidet sich leicht (siehe dazu den Quellcode auf den Datenträgern der Anlage F). Schematisch stellt sich der einfache Aufbau des Servers wie folgt dar.

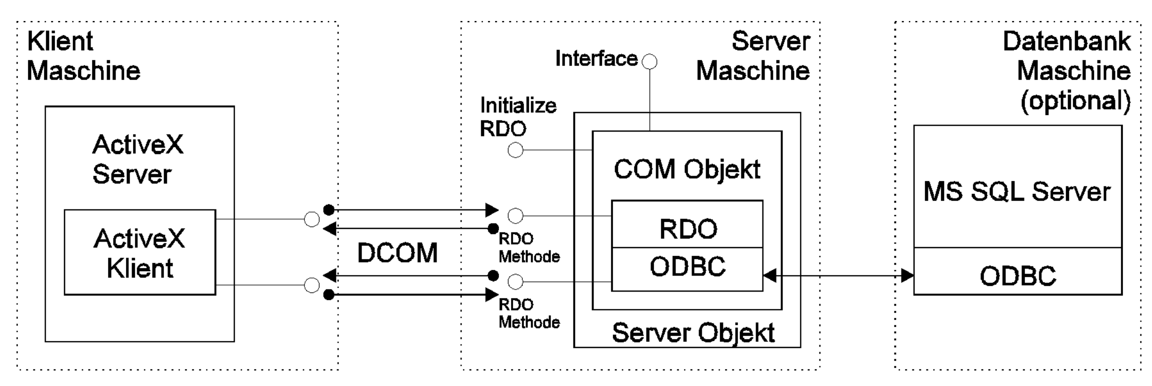


Abb. 39: Anwendungsrealisierung unter Visual Basic und COM.

Wenn der Server also nur die Aufgabe der Objektinitialisierung hat, drängt sich die Frage nach dessen Notwendigkeit auf. Da, aufgrund der Ortstransparenz von COM, das RDO Objekt nicht auf dem Klientensystem installiert sein muß, könnte der ActiveX Klient das Objekt direkt selbst instantiieren, initialisieren und dessen Methoden aufrufen. Diese Annahme ist aus zwei Gründen jedoch nicht korrekt. Zum einen ist das Datenobjekt (sei es nun RDO oder DAO) selbst nicht als eigenständiger Server, also weder lokal noch entfernt, sondern als In-Process Server implementiert. Zum anderen, und dieser Grund stellt auch gleichzeitig die zweite Aufgabe des Servers dar, kann nicht davon ausgegangen werden, daß die ODBC-API auf dem Klientensystem installiert ist. Der realisierte Server stellt die ausführbare „Hülle" für das In-Process Serverdatenobjekt zur Verfügung und greift dabei auch auf den ODBC-Klienten auf seinem System zu.

In der vorliegenden Implementierung übernimmt der Server direkt die Authentifizierung des Anwenders, indem die Konteninformationen direkt in den Code implementiert sind. Dieses Vor-

gehen mußte gewählt werden, um den Server als Dienst implementieren zu können. Dies wäre bei der Verwendung eines interaktiven Authentifizierungsdialoges nicht möglich. Ein *Dienst* ist das WinNT Gegenstück zu einem Daemon. Er ist auch dann aktiv, wenn kein Anwender an dem WinNT Betriebssystem angemeldet ist, und kann einen vorgegebenen Anwender impersonifizieren um so vorgegebene Rechte auf dem System zugewiesen zu bekommen. Eine Implementierung des Servers als eigenständiges Programm, wie sie auch bei dem JAVA-Server erfolgt ist, kann natürlich einen Authentifizierungsdialog hervorbringen.

Um eine Anwendung als WinNT-Dienst zu registrieren ist wie folgt vorzugehen [MS396]:

1. <Drive>:\<ResKitDir>\INSTSRV.EXE
<ServiceName><Drive>:<TargetDir>\DCOMServer.EXE
2. In dem Registry-Zweig \SYSTEM\CurrentControlSet\Services ist der folgende Eintrag einzufügen:
Value Name: Application
Data Type: REG_SZ
String: <Drive>:\<Target Dir>\DCOMServer.EXE

Die Konfiguration als Dienst hat den Vorteil, daß auf dem Server kein Anwender angemeldet sein muß und er damit ohne Interaktion durch einen Anwender betrieben werden kann. Dies ist bei der Implementierung des JAVA-Servers zur Zeit noch nicht möglich.

Auf die vollständige Implementierung des Klienten wurde verzichtet, da hier keine Erkenntnisse zu erwarten sind. Das Datenobjekt (RDO oder DAO) wird ohne Verlust der Mächtigkeit dem Klienten zur Verfügung gestellt. Dies geht sogar so weit, daß Komponenten für die Visualisierung (wie z.B. ein Listenelement) direkt an das Datenobjekt gebunden werden können. Damit entfällt jeglicher Programmieraufwand für die Manipulation der Daten, lediglich die GUI muß erstellt werden.

Auf der Seite des Klienten wird ein sogenanntes „ActiveX Executable” realisiert. Ein OCX für die Darstellung in einem WWW-Browser kann analog generiert werden. In einem solchen Projekt müssen folgende Konfigurationen beachtet werden:

Im Klassenmodul wird die Instantiierungseigenschaft auf „MultiUse” gesetzt. In einem Klassenmodul werden die Servermethoden implementiert. Folgende Projekteigenschaften müssen zusätzlich gesetzt werden:

Project Type:	ActiveX EXE	(Create out-of-process ActiveX component)
Project Name:	DCOMServer	(Identifiziert COM-Objekt in der Registry)

Die Option „Nativen Code Kompilieren” sollte deaktiviert werden um Portabilität zu gewährleisten. Weiter sollte die „Versionskompatibilität“ aktiviert werden, um die GUI jederzeit aktuell zu halten. Nachdem eine ausführbare Datei erstellt wurde, kann durch die Versionskompatibilität auf eben diese verwiesen werden.

Der Server kann mit Hilfe der Installationsroutinen zur Verteilung vorbereitet werden. Dabei sind keine Angaben mehr nötig. Weitere ActiveX-Komponenten werden nicht verwandt. Bei den Abhängigkeiten der Installationsumgebung ist darauf zu achten, daß die Datei STDOLE2.TBL eingebunden ist. Die Installation muß nicht ist als „Shared Component“ durchgeführt werden, da die Kommunikation von (D)COM gehandhabt wird.

Der Klient wird als „Standard EXE“ erstellt. Als Projektreferenz wird der oben generierte Server angegeben. Durch die Fähigkeit des ActiveX-Servers, sich selbst zu registrieren, erscheint er unmittelbar nach Erstellung in der Auswahl der aktiven Komponenten (bei Remote OLE Automation in VB4.0 wäre eine manuelle Registrierung durch Starten des Servers nötig).

Der Kern der Objektreferenzierung im Klienten stellt sich wie folgt dar:

```
Option Explicit
Public oServer As New DCOMServer.clsDCOMServer
```

Die Methoden des Serverobjektes können, wie bereits in der einführenden Darstellung des Codesegments des Klienten erläutert, sofort ausgeführt werden. Beim Beenden des Klienten durch den Anwender muß das Objekt wieder freigegeben werden.

```
Private Sub Form_QueryUnload(Cancel As Integer, _
                             UnloadMode As Integer)
  Set oServer = Nothing
End Sub
```

Das Objekt `Nothing` ist dabei das MS Visual Basic Pendant zum NIL Zeiger in C++.

4.3.6 Bewertung der Implementierung

Die hier vorgestellte Implementierung zeichnet sich insbesondere durch ihre Einfachheit und der damit verbundenen sehr kurzen Entwicklungszeit aus. Nachdem die Konfiguration von COM selbst aufwendig und durch Microsoft noch unzureichend dokumentiert ist, ist die anschließende Verwendung dieser Middleware gerade durch eine Hochsprache wie Visual Basic, auch durch Einsatz des Dispinterfaces (vgl. Kapitel 3.2.3.2.2.2), denkbar einfach. Aufgaben wie IDL Spezifikation werden vollständige transparent.

Im Gegensatz zu der vorgestellten JAVA/CORBA Implementierung bietet dieser Aufbau die Möglichkeit der Gestaltung eines 2-Tier-Modells [STE96], in dem die vermittelnde Schicht entfällt. Dies hat zwei Gründe: zum einen kann eine ActiveX-Komponente direkt mit dem nativen ODBC-Klienten auf dem Klientensystem kommunizieren. Zum anderen unterliegt das ActiveX-Modell keinen Sicherheitsrestriktionen in der Form, daß nur eine Kommunikationsverbindung zum ursprünglichen WWW-Server aufgebaut werden kann. Damit ist, über den lokalen ODBC-Klienten, eine direkte Kommunikation mit der Datenbank möglich. Dieses Vorgehen wurde hier jedoch nicht gewählt, um nicht die Installation einer ODBC-Schnittstelle auf dem Klienten zu erzwingen. Dadurch, daß im 3-Tier Aufbau die Datenbankschnittstelle als mittlere Schicht auf einem Server liegt, ist die resultierende ActiveX-Komponente auf dem Klienten sehr klein [STE96].

Ein weiteres bemerkenswertes Detail und starker Vorteil gegenüber dem CORBA-Modell ist, daß die Authentifizierungsmechanismen des MS SQL Servers, ohne Modifikationen, zur sichern Authentifizierung des Klienten im Internet verwendet werden können. Da unter WinNT eine Integration der ACL des Betriebssystems selbst, und der des MS SQL Servers möglich ist, kann über diesen Mechanismus eine homogene Benutzerverwaltung, sowohl für den Zugriff auf den WWW-Server als auch auf den MS SQL Server realisiert werden. Zusätzliche und unterstützende Protokolle und Mechanismen wie SHTTP und SSL sind dabei unnötig.

Nicht zuletzt die einfache Integration des RDO, und die damit verbundene Möglichkeit von „Prefetching“ und „Offline Cache“, machen dieses Modell für einen Einsatz in Intranet variabler Bandbreite geeignet und dem implementierten JavaIDL-Modell überlegen.

4.4 Vergleich von ActiveX und JAVA

4.4.1 Durchsatzanalyse

Die Bewertung der Leistungsfähigkeit der beiden realisierten Modelle basiert auf der Betrachtung von vier Teilaspekten, die zusammen ein Gesamtbild zur Bewertung liefern. Um die Kapazität der verschiedenen Verbindungsarten klassifizieren zu können, wird hier, neben der theoretischen Übertragungsrate der Verbindung, die „Round Trip Time (RTT)“ eines Ping-Paketes für die Bewertung herangezogen (vgl. Anhang C). Diese vereinfachende Einschränkung wird in Kauf

genommen, da eine effektive Rate der Datenübertragung der Medien von zu vielen Aspekten beeinflußt wird (siehe dazu auch den Anhang C). Gerade weil in dem implementierten Beispiel sehr viele Methodenaufrufe keine, oder nur wenige Daten übertragen (vergleiche z.B. „TriggerEvent“ Abb. 35), ist hier die Klassifizierung durch die Verzögerungszeit des sehr kleinen Ping-Paketes ein guter Kompromiß.

Die im folgenden dargestellten Ergebnisse basieren auf den Messungen im Anhang C.

Zunächst ist die Zeit von Interesse, die ein Klient für das Anbinden an ein Serverobjekt benötigt. Darunter fallen das Suchen und Finden der Serverkomponente und die Etablierung der Kommunikationsverbindung selbst. Die Messung ergibt für die JAVA-Implementierung eine verhältnismäßig konstante Zeit für den Verbindungsaufbau, die damit unabhängig von der Qualität des Transportmediums ist. Ein durchgeführter Scan (Sniffing) der übertragenen Daten auf Paketebene bestätigt die Vermutung, daß die gemessenen Zeiten von ca. 25s nicht von Datenübertragungen herrührt.

Die verwendete Version von JavaIDL unterstützt nur einen sehr einfachen Namensdienst, bei dem der Server seine Objektreferenz dem ORB publiziert, die dann vom Klienten angefragt werden kann. Die gemessene Zeit wird also für die Instantiierung des Skeleton benötigt. Dem gegenüber ergeben sich für den Verbindungsaufbau bei COM, fallende Werte für schnellere Verbindungen. Der Scan der übertragenen Pakete zeigt hier, daß während der gesamten Aufbauphase Verbindungsdaten ausgetauscht werden, was bei schnelleren Verbindungen zu kürzeren Etablierungszeiten führt. Bei dem für das Intranet der GTZ relevanten Wert der physikalischen Übertragungskapazität von 9600bps liegt die Zeit für den Verbindungsaufbau im COM bei ca. 5s. Die Stärke von COM gegenüber JavaIDL liegt aber klar in dem Bereich der schnelleren Verbindungen. In dem folgenden Diagramm wird die Zeit dargestellt, die für den Verbindungsaufbau benötigt wird. Auf der Abszisse wird die Kapazität der Kommunikationsverbindung aufgetragen, die sich aus den gemittelten Ergebnissen der `Ping()` Operation ergeben.

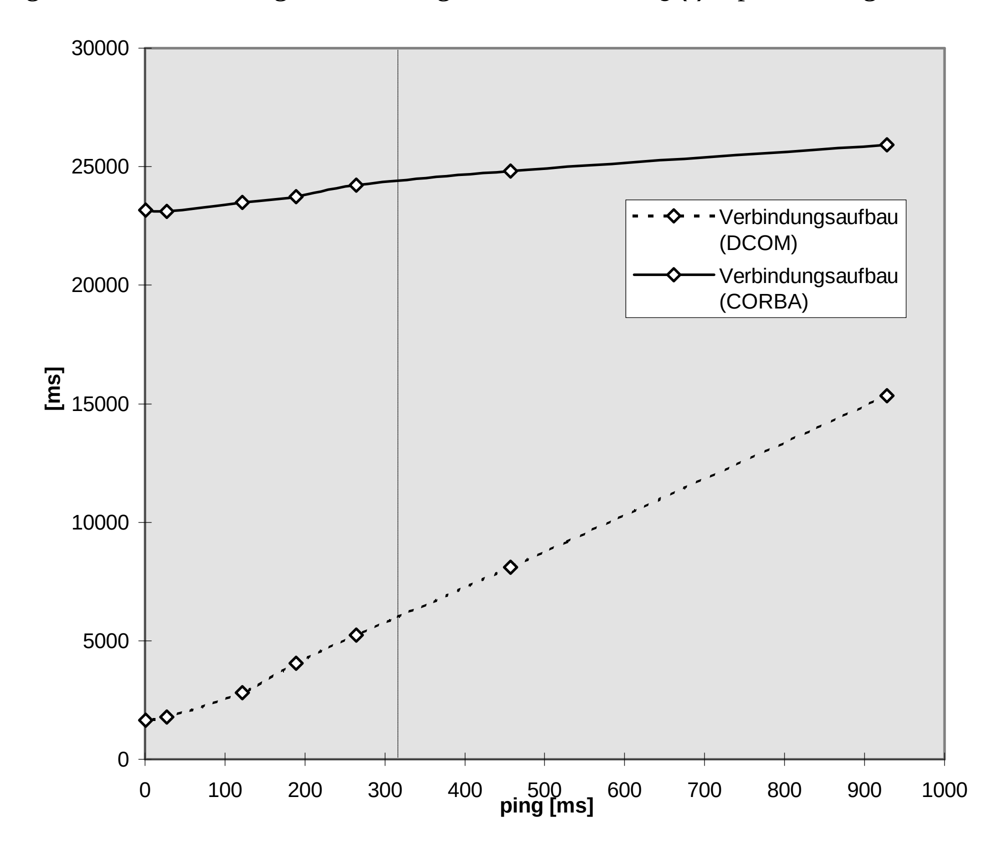


Abb. 40: Vergleich der Verbindungsaufbaugeschwindigkeit von COM und JavaIDL.

Der zweite Aspekt in der Betrachtung der Leistungsfähigkeit der Modelle ist die Zeit, die für den Aufruf und die Ausführung einer einzelnen Methode benötigt wird. Hier bieten beide Modelle in der Darstellung eine stetige, lineare Funktion in Abhängigkeit von der Leistungsfähigkeit der Verbindung. Mit abfallender physikalischer Übertragungskapazität wachsen dabei die Vorteile der JavaIDL Implementierung. Dies kann nur auf ein größeres Datenvolumen zurückzuführen sein, daß bei COM für die Ausführung übertragen werden muß. Dies ist auch nicht weiter verwunderlich, da die Kombination von ActiveX und MS Visual Basic auf einer sehr viel höheren Abstraktionsebene erfolgt, die zwangsläufig Overhead verursacht (vgl. Kapitel 3.2.3.2.2.2). Nicht zuletzt erfolgt die Übertragung von COM verschlüsselt. Der „Data Encryption Standard (DES)" generiert zwar aus 64-Bit-Klartext Paketen auch 64-Bit Chiffretext, verursacht also für den Transport der Daten keinen erhöhten Datenverkehr, der Algorithmus muß aber zweimal durchgeführt werden [TAN97]. Bei 9600bps Leitungsgeschwindigkeit liegen die Zeiten für einen Methodenaufruf bei ca. 350ms für COM und bei ca. 200ms für JavaIDL.

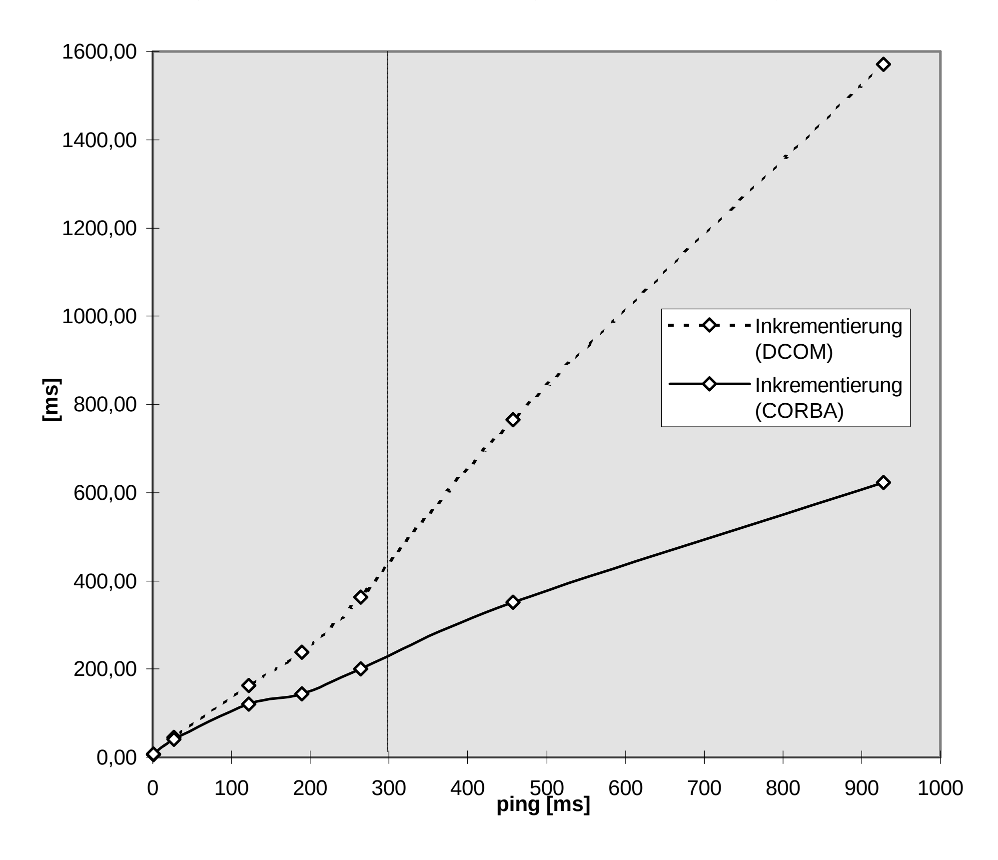


Abb. 41: Vergleich der Ausführungsgeschwindigkeit von COM und JavaIDL.

Durch diese gegensätzlichen Ergebnisse im Vergleich der Aufbau- und Ausführungsgeschwindigkeit stellt sich die Frage, bei welcher Kombination von Anzahl von Operationen und physikalischer Übertragungskapazität das jeweilige Modell vorzuziehen ist.

Um eine Vergleichbarkeit der Messungen von COM und JavaIDL zu erreichen, wurde in dem folgenden Diagramm eine Break-Even Analyse durchgeführt. Darin wurden sowohl die Zeit für den Verbindungsaufbau als auch die Verarbeitungsgeschwindigkeit für die Ausführung einer Operation zur Inkrementierung eines Integers einbezogen. Innerhalb des Versuches wurden auch andere mathematische Operationen durchgeführt, für die sich aber sehr ähnliche Meßwerte ergaben. Im folgenden Diagramm wird demnach dargestellt, bei wievielen Operationen (Transaktionen) die Auswahl der Modelle zu einem gleichen Ergebnis führt. Diese Break-Even Analyse gibt also die Anzahl von sofort aufeinanderfolgenden Methodenaufrufen an, für welche die Verwendung der beiden Modelle indifferent ist. Für sehr langsame Verbindungen empfiehlt sich der Einsatz von JavaIDL schon bei wenigen Aufrufen, bei schnellen Verbindungen hat COM

deutliche Vorteile. Bei der Übertragungsgeschwindigkeit von 9600bps liegt der Break-Even Wert bei ca. 100 Aufrufen.

Die berechneten Werte ergeben sich aus folgender Vorschrift (vgl. dazu [COL89]):

$$x_{\text{Break-Even}}(RTT) = \frac{Aufbau_{\text{JavaIDL}}(RTT) - Aufbau_{\text{COM}}(RTT)}{Transaktion_{\text{COM}}(RTT) - Transaktion_{\text{JavaIDL}}(RTT)}$$

Daraus ergibt sich die folgende Darstellung:

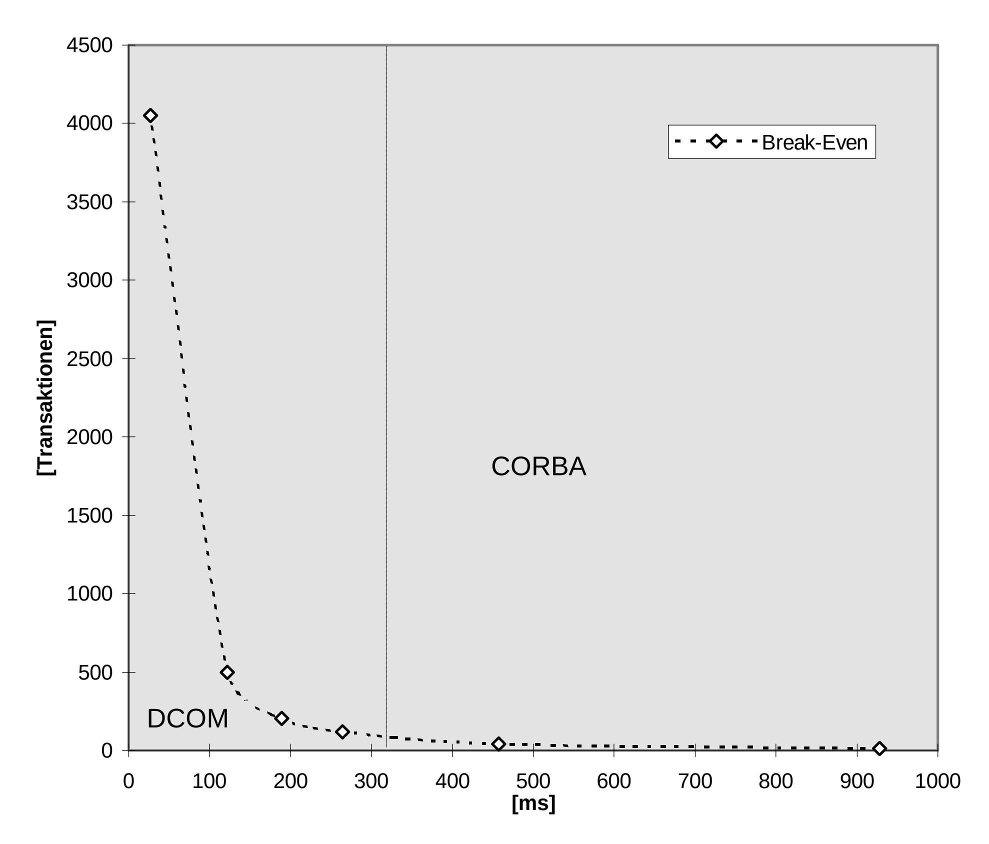


Abb. 42: Break-Even Analyse der Transaktionszahlen.

Für COM ergibt sich dadurch bei 9600bps ein errechneter Grenzwert von 2,9 Transaktionen/s (bzw. 120 aufeinanderfolgenden Transaktionen nach dem Verbindungsaufbau), ab der diese Middlewareplattform nicht mehr eingesetzt werden sollte. Für das JavaIDL-Modell beträgt der entsprechende Wert 1,8 Transaktionen/s, ab dem sich der Einsatz dieses Modells empfiehlt. Dieser Grenzwert kann leicht in der Beispielimplementierung durch einen dynamischen Datenbestand erreicht werden, da in diesem Fall schon die Aktualisierungshinweise (siehe „Trigger-Event“ Abb. 35), die vom JAVA-Server an den Klienten zurückgegeben werden, eine größere Transaktionszahl ergeben können.

Bei dieser Bewertung, die gerade bei niedrigen Übertragungskapazitäten den Einsatz von JavaIDL stark favorisiert, ist aber noch ein letzter Aspekt zu beachten. Im Gegensatz zu ActiveX muß in dem JavaIDL Modell der Bytecode zur Laufzeit auf den Klienten übertragen werden. Im folgenden muß also zunächst untersucht werden, wie groß eine derartige objektorientierte, verteilte, modulare Anwendung ist. Aufgrund der Größe der Objekte können Aussagen über die erforderliche Zeit für die Übertragung der Objekte bei verschiedenen gängigen Netztechnologien getroffen werden.

Die folgende Tabelle zeigt die Übertragungszeiten (in Sekunden) von Daten unterschiedlicher Größe über unterschiedliche Medien. Um die Medien vergleichbar zu machen wurde hier die maximale Datenrate (in Bits pro Sekunde) als Rechengrundlage verwendet.

Größe [KB]	*Modem 2400bps*	*Modem 4800bps*	*Modem 9600bps*	*Modem 14,4Kbps*	*Modem 28,8Kbps*	*ISDN 64Kbps*	*Ethernet 10Mbps*
1	3,41	1,71	0,85	0,57	0,28	0,13	0,0008
5	17,07	8,53	4,27	2,84	1,42	0,64	0,0041
10	34,13	17,07	8,53	5,69	2,84	1,28	0,0082
50	170,67	85,33	42,67	28,44	14,22	6,40	0,0410
100	341,33	170,67	85,33	56,89	28,44	12,80	0,0819
200	682,67	341,33	170,67	113,78	56,89	25,60	0,16
500	1706,67	853,33	426,67	284,44	142,22	64,00	0,41

Tab. 20: Ladezeiten über verschiedene Medien [SVE96].

Diese Werte entsprechen nicht dem effektiven Durchsatz des jeweiligen Übertragungsmediums. Dieser unterliegt für eine zuverlässige Aussage aber zu vielen Einflüssen, wie zum Beispiel Komprimierungsverfahren, Komprimierbarkeit der Daten, Last und Hardware. Ein 28800 Baud Modem liefert zum Beispiel in der Praxis, je nach Leitungsqualität, bis zu etwa 3.5KByte/s [HAL96]. Weiter müßte bei dieser vollständigen Betrachtung auch der Overhead des Transportprotokolls TCP/IP mit ca. 30% der Übertragungskapazität berücksichtigt werden [SVE96]. Als Vergleichswert bietet das Intranet der GTZ auf den internationalen Verbindungen eine hohe durchschnittliche Übertragungsrate von 6 KBytes/s. Dieser gute Wert wird durch Kanalbündelung, hohe Paketgrößen und starke Komprimierung erreicht. Durch Verwendung des theoretischen Durchsatzes kann aber in jedem Fall eine obere Abschätzung der Übertragungszeit vorgenommen werden.

Diese vorgestellte einfache und nicht aufwendige Implementierung des Datenbankklienten umfaßt bereits ca. 60KByte. Dazu addieren sich bis zu 200KByte an Klassen des JavaIDL. Solange diese Klassen nicht in die VM der Klienten integriert sind, verlängert sich die Ladezeit entsprechend. Im relevanten Umfeld von 9600bps erreicht man dabei eine abgeschätzte Ladezeit von ca. 3,5 Minuten (im Intranet der GTZ sinkt der Wert auf eine knappe Minute). Erschwerend kommt noch hinzu, daß die Vielzahl von kleinen Dateien einen wesentlich höheren Protokolloverhead verursachen, als dies bei einer einzelnen (entsprechend größeren) Datei der Fall wäre. Die Ladezeit der HTML-Seite muß nicht berücksichtigt werden, da diese für beide Technologien eine ähnliche Größe hat.

Im Vergleich dazu hat ActiveX eine integrierte Versionskontrolle, die sowohl für die VM selbst als auch für die erstellten Komponenten greift. Die einzelnen Dateigrößen sind zwar beträchtlich - so hat die VM ein Größe von 1,3MB, die RDO Bibliothek eine Größe von 400KB und der realisierte, sehr schmale Datenbankklient umfaßt 20KB - was in einer erheblichen initialen Ladezeit resultiert. Ist eine ActiveX-Komponente aber einmal geladen worden, muß diese vorhandene Version nicht wieder über das Netzwerk übertragen werden.

Als Ergebnis kann also besonders für langsame Verbindungen festgehalten werden, daß der Vorteil von JavaIDL leider durch die Ladezeiten in den Hintergrund gedrängt wird. Zwar wird an der Kombination von Klassen in einer einzelnen Datei gearbeitet, und ein Zwischenspeichern des JAVA Bytecodes auf dem Klienten durch Zusatzprodukte ist möglich, die Antwortzeit des Modells selbst ist aber nicht zu akzeptieren.

4.4.2 Beurteilung und Performanzanalyse des Datenzugriffes

Im Vergleich der Performanz der Datenobjekte wurde auf die Messung über verschiedene Übertragungskanäle verzichtet, da hier die Last durch den Datenverkehr erzeugt wird und für den Datenaustausch in beiden Fällen die ODBC-Bibliothek verwendet wird. Als repräsentative Messung kann so exemplarisch ein 10BASE-T Ethernet als Kommunikationsplattform gewählt werden. Das Netzsegment wurde isoliert betrieben, um etwaige Störungen durch parallelen Zugriff auf das Medium zu vermeiden. Die Meßwerte, die im Anhang C ermittelt wurden, zeigen, daß das RDO eine marginal höhere Zeit für den Verbindungsaufbau benötigt. Der zweite

interessante Wert ist die Erstellung des eigentlichen RecordSets, als die lokale, virtuelle Repräsentation der Datenbasis auf dem Klienten. Hier wird für das RDO eine doppelt so hohe, gemittelte Aufbauzeit gemessen.

Grundsätzlich sind die beiden Systeme bei der Performanzanalyse in der Größenordnung der gemessenen Werte vergleichbar (vgl. Tab. 21). Die etwas schlechteren Werte im Einsatz von RDO resultieren aus der höheren Leistungsfähigkeit des Datenobjektes. Die markanten Merkmale der beiden Mechanismen werden in der folgenden Tabelle dargestellt. Ein für die vorgestellte Implementierung sehr wichtiges Merkmal ist das Fehlen des „aktiven RecordSets“ in der JDBC-ODBC Umgebung. Dies hat zur Folge, daß die Benachrichtigung über die Veränderung des Datenbestandes aufwendig vom MS SQL Server über den JAVA-Servern an den JAVA-Klienten übermittelt werden muß. Das RDO bietet hierfür einen fertigen Mechanismus, der keinen zusätzlichen Aufwand für die Implementierung erforderlich macht. Die RDO-Implementierung macht die Verwendung von Triggern auf der Seite des MS SQL Servers überflüssig (siehe dazu auch Abb. 35).

Eigenschaft	***JDBC-ODBC Bridge***	***RDO***
Veröffentlichte Schnittstelle	JAVA Packages	C Funktion Prototypen
Interne Architektur	JAVA Objekte auf C++ Klassen	C Strukturen/C++ Klassen
Objekt Manipulation	JAVA Objekte	Handles
Klientencode	JAVA	C/C++/VB etc.
Transaktionskontrolle	Ja	Ja
Transaktionskontrolle über mehrere Verbindungen	Nein	Ja
Passiver Offline Cache	Nein	Ja
Aktiver Offline Cache	Nein	Ja
Recordset Typen	Statisch	Snapshot, Statisch, Dynamisch
Stored Procedures	Ja	Ja
Cursor-Unterstützung	Tabelle	Tabelle / Stored Procedure
Scrollable Cursors	Nein	Ja
Abfragen als Methoden	Nein	Ja
ODBC Bookmarks	Nein	Ja
Asynchrone Abfragen	Durch JAVA Threads mit erheblichem Aufwand	Ja

Tab. 21: Vergleich von JDBC-ODBC Brücke und RDO.

Diese Tabelle spiegelt auch den Entwicklungsaufwand der Implementierung wieder. Dadurch, daß das RDO für die Verteilung von Datenbankanwendungen wichtige Merkmale wie dynamischen Zwischenspeicher, Cursor-Unterstützung auf Ebene von gespeicherten Prozeduren und Abfragen als Methoden unterstützt, ist in diesen Punkten keine Implementierung erforderlich. Die Entwicklungszeit und die damit verbundenen Kosten sind minimal.

Demgegenüber steht die sehr aufwendige Implementierung der RecordSet-Klasse in der Beispielimplementierung auf Basis von JavaIDL. Die Kombination von dieser Klasse mit der separaten C++-Implementierung des TriggerEvents stellt die Voraussetzungen für einen aktiven, in JAVA realisierten, Zwischenspeicher auf der Seite des Datenbankklienten dar. Dabei ist die Verwendung von gespeicherten Prozeduren für die Realisierung des TriggerEvents ein entscheidender Optimierungsschritt im Modelldesign der vorgestellten Implementierung. Aufgrund der sehr unzureichende Performanz der heute verfügbaren VMs, kann nur durch Verlagerung der Rechenleistung auf den Datenbankserver eine Beschleunigung bei der Verarbeitung von Abfragen erreicht werden [DIC97]. Dieser Schritt ist im RDO-Modell nicht erforderlich. „Inwiefern sich diese Situation mit zukünftigen Compilergenerationen (Interpreter

mit JIT-Compiler oder der angekündigte schnelle Host-Spot-Compiler) ändert, bleibt abzuwarten [DIC97].“ Diese neuen Technologien werden im folgenden Kapitel vorgestellt.

5 Auswertung und Ausblick auf weitere Entwicklungen

5.1 Verfügbarkeit

Die Entwicklungen im Bereich der Middlewareplattform COM sind vielschichtig. In Kooperation mit Microsoft hat die Software AG im Spätsommer 1997 die erste Betaversion von DCOM FTE (DCOM for the Enterprise) für die Betriebssysteme SOLARIS, LINUX und Digital UNIX vorgestellt. Wenn DCOM FTE das COM auf diesen Plattformen unterstützt, ist das allerdings nicht gleichbedeutend damit, daß binäre COM-Komponenten auf UNIX lauffähig sind, da keine INTEL Emulation für binäre Komponenten realisiert wurde. Vielmehr leistet DCOM FTE Kompatibilität auf Quelltextniveau.

Die Laufzeitumgebung besteht aus einigen Shared Libraries, die einen Teil der NT Basisfunktionen emulieren. Dazu gehören Dateifunktionen, Threads, Semaphore und Funktionen für die Registry-Datenbank. Zwei Daemons (*ntd* und *rpcss*) bilden eine Untermenge des Win32-Teilsystems und den „Service Control Manager (SCM)“ nach [WIT97]. Somit ist für die Portierung von Komponenten keine Modifikation an der Middleware erforderlich. Die Portierung der Threads für das öffentliche Betriebssystem LINUX ist noch nicht realisiert, da die Unterstützung von Threads unter LINUX noch nicht abgeschlossen ist.

Wichtige Werkzeuge von WinNT wurden nachgebildet. So steht zum Beispiel der Registry-Editor „RegEdit“ als Motif-Anwendung zur Verfügung. Unter LINUX fehlt RegEdit als GUI-Anwendung, da die nötigen X11-Bilbliotheken nicht threadsicher programmiert sind. Wichtige Werkzeuge wie der MIDL-Compiler sind auf allen Plattformen vorhanden. Einige MS Windows Spezifika, wie zum Beispiel das „Apartment Threading Model“ wurden nicht implementiert. Schwierigkeiten treten auch bei der Konvertierung von ASCII/ANSI Codes in UNICODE auf. Auch hat die SAG die Authentifizierung von Komponenten noch nicht fertiggestellt, was eine große Sicherheitslücke im COM öffnet. Der gesamte Sicherheitsmechanismus unter WinNT kann unter den neuen Systemen nicht verwendet werden [WIT97].

Ob COM sich durchsetzen kann, hängt unter anderem von der Anzahl der unterstützen Plattformen ab, da eine heterogene Umgebung im allgemeinen nicht ausschließlich aus MS Windows Systemen besteht. Mittlerweile sind Implementierungen für alle Versionen von WinNT erhältlich. Für Win95 ist DCOM als Modul frei verfügbar und für MS Windows 98 als Bestandteil angekündigt. DCOM soll für den Apple Macintosh, der bereits COM teilweise unterstützt (siehe Anhang D), noch in 1997 folgen [WIT97]. Schwierigkeiten bei dieser Umsetzung ergeben sich vor allem durch die Integration der Win32-API im COM, was eine Abbildung auf andere Plattformen erheblich erschwert (nicht zuletzt auch weil die API nicht vollständig durch MS dokumentiert ist).

Zur Zeit unterstützen folgende Betriebssysteme DCOM Anwendungen [MS965]:

Betriebssystem	*DCOM*
MS Windows NT Server	ja
MS Windows NT Workstation	ja
MS Windows 95[24]	ja
LINUX	Betaversion
SOLRAIS	Betaversion
Digital UNIX	Betaversion
Apple Macintosh	angekündigt

Tab. 22: Verfügbarkeit von MS DCOM.

[24] Zusatz “DCOM für Win95” muß installiert sein.

Obwohl ActiveX eine sehr neue Technologie ist, bedeutet die Adaption von OLE 2.0 in das Komponentenmodell, daß schon jetzt eine nicht einzuholende Verbreitung dieser Technologie aus der Verbreitung von Personalcomputern mit MS Windows Betriebssystem resultiert.

Dem gegenüber steht die Frage nach der Verfügbarkeit und Unterstützung von JAVA. Das JDK ist offiziell auf den folgenden Plattformen verfügbar.

Betriebssystem	***JDK Version***
Microsoft Windows 95 / NT 4.0	1.1.4 Final[25]
SUN Solaris 2.4, 2.5 SPARC	1.1.4 Final
SUN Solaris 2.5 x86	1.1.4 Early-access
Portierungen von Drittanbietern[26] • AIX • AmigaOS • BeOS • diverse UNIX und Derivate • HP-UX • MacOS • OS/2	Verschiedene Browser, VMs und JDKs, zum Teil noch als Betaversionen.

Tab. 23: Verfügbarkeit des JDK.

Einen bedeutenden Vorteil hat jedoch JAVA gegenüber ActiveX. Dadurch, daß die JAVA VM plattformunabhängig ist, und nicht wie ActiveX Teile des Betriebssystems des Klienten verwendet, ist eine Implementierung und Portierung dieser VM wesentlich einfacher. Als Konsequenz daraus unterstützen heute bereits eine Vielzahl von WWW-Browsern (wie Netscape Navigator, SUN HotJava, MS InternetExplorer und IBM WebExplorer) JAVA, von denen aber nur der MS InternetExplorer ActiveX-Komponenten ausführen kann. Darüber hinaus hat SUN für das 3. Quartal '98 seinen JAVA Mikroprozessor angekündigt. Das Herz des sogenannten „mircoJava 701" ist ein verbesserter „picoJava 2.0"-Kern, ein Hochleistungskern, der für die Ausführung von nativem JAVA Code optimiert ist, aber auch die Ausführung von C-Code unterstützt. Diese Entwicklung ist Teil der Tendenz, hin zum Internetcomputer; ein Personalcomputer, der vor allem durch seine spartanische Hardwareausstattung und dem damit verbundenen niedrigen Preis auffällt, plattformunabhängigen JAVA Code verarbeiten soll und für den Endverbraucher eine Alternative zur teureren Workstation bietet. Die Etablierung dieser „Network Computer (NC)" wird nicht zuletzt auch von der Einführung der sogenannten „Just in Time (JIT)" Compiler abhängen. Diese Compiler verändern die Rolle der VM leicht, indem sie Bytecode direkt in nativen, plattformabhängigen Code umwandeln. Dadurch wird die VM von nativen Aufrufen entlastet. Durch diesen Vorgang kann die Ausführungsgeschwindigkeit stark gesteigert werden [MOR97].

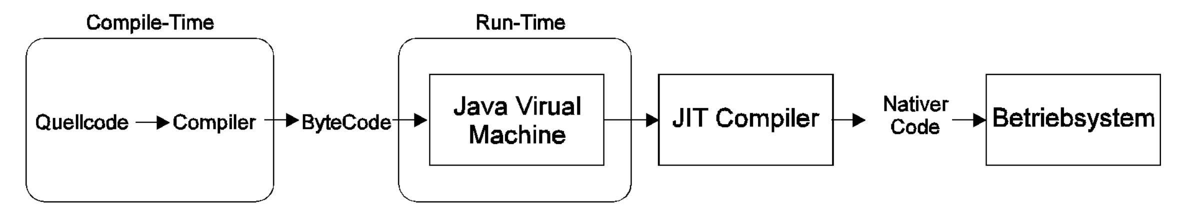


Abb. 43: JIT-Compiler unterstützen die VM von JAVA [MOR97].

Eine JAVA-Klasse wird dabei durch die VM in den Arbeitsspeicher geladen, indem eine sogenannte VTable (Virtual Table, die Bezeichnung entspricht dem C++ Pendant, inhaltlich besteht aber nur geringe Übereinstimmung, vergleiche dazu auch [MOR97]) ausgelesen wird, die Zeiger auf alle Methoden der Klasse enthält. Dies ist ein ähnliches Vorgehen, wie bei den

[25] JDK scheint zwar auf WinNT 3.51 lauffähig zu sein, ist aber nicht offiziell freigegeben.

[26] Für eine vollständige Liste der Drittanbieter siehe http://java.sun.com/cgi-bin/java-ports.cgi.

VTables der COM-Komponenten. Bei Verwendung eines JIT-Compilers wird jede Adresse innerhalb der VTable, die ja auf Bytecode verweist, durch die Adresse des JIT-Compilers selbst ausgetauscht. Bei einem Methodenaufruf der VM über die VTable generiert nun der JIT-Compiler - just in Time - nativen Binärcode. Das Verfahren basiert also auf der Verarbeitung der einzelnen Methoden, so daß nur die Methoden kompiliert werden, die auch tatsächlich aufgerufen werden. Dadurch, daß JIT-Compiler nach der vierten Schicht des JAVA Sicherheitsmodells ausgeführt werden, verursachen sie kein weiteres Sicherheitsrisiko; sie operieren auf Bytecode, der schon auf Sicherheit überprüft ist [MOR97].

Hier nähern sich beide Technologien in zwei Punkten an: Zum einen unterstützt JAVA, wie auch ActiveX, mit dem JIT-Compiler eine Form von „ausführbarem“ Binärcode, der klare Vorteile in der Ausführungsgeschwindigkeit bietet. Zum anderen erzielt die Verwendung dieses Compilers durch das selektive Laden von binären Methodenkomponenten einen Performanzgewinn (vgl. Kapitel 2.3.2). Auch dieses Verfahren ist in der ActiveX-Umgebung durch In-Process Server oder DLLs bekannt.

Die folgenden, wichtigen Entwicklungen in der JAVA-Technologie sind erforderlich, um JAVA als Alternative zu ActiveX zu etablieren, und um damit die zukünftige Verfügbarkeit von JAVA zu gewährleisten:

1. Die Einführung von „Internet Foundation Classes“ [MOR97], welche die Wiederverwendung von vorgefertigten Objekten ermöglichen, die in JAVA geschrieben und damit ebenfalls plattformunabhängig sind. Diese werden eine Basis für kommerziell vermarktbare Komponenten bieten.
2. Wachsende JAVA-Unterstützung auf Serverseite. Dadurch wird eine Verteilung von netzwerkzentrierten Anwendungen auf Klienten vereinfacht, indem eine einheitliche Entwicklungsumgebung für die gesamte Systementwicklung zur Verfügung gestellt wird.
3. Die vollständige Integration des IIOP in JAVA wird sowohl der Sprache als auch der Plattform neue Einsatzmöglichkeiten eröffnen. Die zur Zeit zur Verfügung stehenden CORBA-Implementierungen sind entweder zu langsam oder, im Vergleich zu COM, (noch) zu teuer, um sich schnell und weit zu verbreiten.
4. Eine flexibleres Sicherheitsmodell wird die rigorosen Einschränkungen aufweichen und damit die Mächtigkeit der Sprache stark erweitern. Nur durch die Möglichkeit des kontrollierten Zugriffs auf das Betriebssystem des Klienten ist JAVA überhaupt als Programmiersprache für Desktop Anwendungen geeignet.

5.2 Vergleich der Middlewareplattformen

Die historische Entwicklung von COM über OLE und OLE 2.0, hin zu ActiveX, verdeutlicht das Fehlen einer übergeordneten und konsistenten Architektur (vgl. Kapitel 3.2.3.2.1). Im Rahmen dieser Entwicklung wurden Pseudostandards verworfen, de facto Standards erschaffen und abschließend zu offenen Standards erklärt. Dagegen verfolgen die Mitglieder der OMG eine gemeinsame „Vision“ einer verteilten, plattformübergreifenden und komponentenbasierten Middleware seit 1989. Diese Architektur ist im „Object Management Architecture Guide“, der 1990 zum ersten Mal veröffentlicht wurde, begründet, und wurde schrittweise zur CORBA Spezifikation erweitert [OMG96]. Dem gegenüber stehen Befürchtungen, daß Microsoft seine Offenheit nach erfolgreicher Einführung und ausreichender Verbreitung seiner neuen Plattform einschränkt. Gegen die CORBA Spezifikation spricht vor allem dessen mangelnde praktische Bedeutung. Implementierungen der einzelnen Dienste sind rar; zum Beispiel ist die vollständige Implementierung des Sicherheitsdienstes durch den Marktführer IONA erst ab diesem Jahr verfügbar (vgl. Kapitel 3.2.1.5). Microsoft bietet eine nahezu vollständige Sammlung von Diensten zum Preis des Betriebssystems WinNT 4.0 von ca. 1000 DM.

Ein weiterer, wichtiger Diskussionspunkt ist die plattformübergreifende Unterstützung. Äußerungen wie „Microsoft wird ohne Entschuldigung sicherstellen, daß ActiveX am besten unter MS Windows arbeiten wird [OMG96]“ von Bob Muglia (Microsoft), stärken nicht das Vertrauen in ActiveX als offenen Standard. Schon allein die Tatsache, daß COM auf Teilen der Win32-API basiert, wird die Portierung auf andere Plattformen erheblich erschweren. Dagegen ist CORBA auf 32-bit MS Windows Systemen, nahezu allen UNIX Versionen, OS/2, OS/400, MacOS, VME, MVS, VMS und einer Zahl von Echtzeit Betriebssystemen verfügbar. Der Standard für DCOM selbst steht erst seit dem 4. Quartal 1996 der Öffentlichkeit zur Verfügung, und die ActiveX Unterstützung von C++ wurde erst in der Version 5.0 von MS C++ komplettiert. Der CORBA 2.0 Standard besteht dagegen abschließend seit Dezember 1994 [OMG96].

Beide Middlewareplattformen zeigen im Methodenaufruf eine ähnliche Performanz, was auch in anderen Meßreihen [ORF97] ermittelt wurde. Konzeptionell ist das CORBA-Modell sauberer, da es direkt von der klassischen Objektdefinition hergeleitet ist. CORBA-Objekte haben eine eindeutige und persistente Referenz und einen Zustand. Dagegen kann auf eine COM-Komponente in einem bestimmten Zustand nach einem Verbindungsabbruch nicht wieder aufgesetzt werden. Die im Zustand enthaltene Information geht verloren [ORF97]. Kritik ist auch am System der Zeiger auf zustandslose Interfaces zu üben, die selbst wieder über persistente COM-Komponenten (Moniker, vgl. Kapitel 3.2.3.2.2.4) realisiert werden. Letztlich hat Mircosoft selbst kaum ein Interesse an einer plattformübergreifenden Kommunikation zwischen Middlewareplattformen. Das IIOP ist nur für CORBA Plattformen verfügbar.

5.3 Eignung der Implementierungen für den Einsatz im Intra- und Internet

Die folgende Gegenüberstellung von JAVA und ActiveX muß inhaltlich auch JavaIDL einbeziehen. Dies ist erforderlich, da ActiveX auf COM als Middleware aufsetzt und dessen Eigenschaften erbt. In dem Vergleich wurden weiter die oben genannten Entwicklungen in JAVA bereits als gegeben angenommen. Dies ist möglich, da sowohl SUN als auch Netscape bereits entsprechende Erweiterungen zum JAVA-Standard angekündigt oder bereits in einer Betaversion veröffentlicht haben. Der Vergleich erstreckt sich nicht über die Middlewareplattformen an sich, sondern auf die in dieser Arbeit vorgestellten Implementierungen und deren Resultate.

Gesichtspunkt	*JAVA*	*ActiveX*
Verfügbarkeit in der Zukunft	Ja (nach Erweiterung)	Ja
Sicherheit	Sandbox	Vertrauen
Verteilte Komponente	Klasse	Objekt
Offener Standard	Ja	Ja
Signatur	Nein	Ja
Versionskontrolle	Nein	Ja
Plattformunabhängigkeit	Ja	Ja (erfordert Win32-API)
Objektcode	Byte	Binär
Objektgröße	zur Zeit noch groß	sehr groß

Tab. 24: Vergleich von JAVA und ActiveX.

In den beiden implementierten Modellen zeigt sich zunächst, daß die Kombination von JAVA und JavaIDL keine „echte“ Verteilung von Objekten zuläßt. Aufgrund der Einschränkungen der VM, durch die ein JAVA-Applet nur eine Verbindung zu seinem ursprünglichen WWW-Server aufbauen kann, muß der Server (oder ein geeigneter Vertreter) auf eben diesem WWW-Server residieren. Diese Einschränkung muß dazu führen, daß ein dedizierter WWW-Server für JavaIDL-Server zur Verfügung gestellt werden muß, um die Last auf mehrere Systeme zu verteilten. Bei der Anbindung eines Intranets an das Internet, kann dieser Server auch auf dem outbound WWW-Server des Unternehmens laufen. Der Server wiederum kann dann, durch ein

Application-Level Gateway, zum Beispiel mit einer Datenbank über ein proprietäres Protokoll kommunizieren.

In dem implementierten Beispiel würde zwischen JAVA-Server und MS SQL Server die Firewall positioniert werden, da der Datenbankserver nicht dem outbound-Netzwerk geöffnet werden sollte. Dieser Restriktion unterliegt ActiveX nicht. Hier kann ein Klient zu jedem beliebigen Server, einschließlich dem Datenbankserver selbst, eine (sichere) Verbindung aufbauen. Dadurch ist innerhalb dieser Architektur auch eine 2-Tier-Lösung möglich. Eine Kommunikation zwischen ActiveX-Komponenten über eine Firewall ist in (D)COM vorgesehen.

Wichtig für den Einsatz in engbandigen Netzen ist die Tatsache, das ActiveX-Komponenten als Binärcode auf den Klienten geladen werden und dort einer Versionskontrolle unterliegen. Die Verwendung von Binärcode hat zwangsläufig Geschwindigkeitsvorteile gegenüber der Verwendung von Bytecode, da der Code direkt ausgeführt werden kann. Dadurch geht allerdings die Plattformunabhängigkeit verloren. Mit der digitalen Signatur einer jeden Komponente ist auch der Schutz des Urheberrechts gekoppelt. Sowohl Schutz von Urheberrecht als auch Versionskontrolle, beides Grundbausteine einer kommerziellen Softwareverteilung, werden vom JAVA-Modell nicht unterstützt. Nachteilig ist auch, daß es zur Zeit keine Möglichkeit unter JAVA gibt, die einzelnen Klassen zu einer einzelnen Datei zu verschmelzen, um so die Ladezeiten zu reduzieren. ActiveX bietet dafür die „Cabinet“ Dateien an.

Der letzte und wichtigste Vergleichspunkt resultiert aus den unterschiedlichen Sicherheitsmodellen der beiden Konkurrenten. Die ActiveX-Technologie setzt ausschließlich auf Vertrauen. Innerhalb eines (nicht hochsensitiven) Intranets stellt dieses Vorgehen keine Probleme dar, sobald aber ActiveX-Komponenten aus dem öffentlichen Internet, durch eine Firewall in das Intranet eines Unternehmens geladen und dort auf einem Klienten ausgeführt werden, unterliegen sie keinerlei Überwachung und Kontrolle. Der Klient, als inbound Computer, hat freien potentiellen Zugriff auf alle Netzwerkressourcen. Obwohl der Autor einer ActiveX-Komponente eindeutig identifiziert werden kann und obwohl Veränderungen am Binärcode der Komponente zuverlässig entdeckt werden können, ist die potentielle Gefahr durch einen mutwilligen Angriff auf das Intranet zu groß. Ein Schaden kann zu einfach verursacht werden und ein Testlauf auf einem abgeschlossenem System ist nicht möglich, da zur Zeit eine selektive Abschottung von ActiveX-Komponenten nicht vorgesehen ist.

5.4 Zukünftige Entwicklungen

Wie auch im Umfeld der traditionellen Programmiersprachen haben sich eine Vielzahl von Programmiersprachen für das Internet entwickelt. Zur Zeit gibt es keinen Grund für die Annahme, daß eine dieser Sprachen in Zukunft eine Monopolstellung innehaben wird, obwohl Verfügbarkeit und Adäquanz für die wechselnden Anforderungen der Internetprogrammierung einige Sprachen bevorzugen. Zum Beispiel ist JAVA sowohl verfügbar als auch generell geeignet, aber neu. Programmierer werden aber offensichtlich zunächst immer den Versuch unternehmen, in ihrer gewohnten und eventuell für das Internet modifizierten Programmiersprache weiterzuarbeiten. Auch Programmieranfänger werden vermutlich eine Programmiersprache höherer Ebenen bevorzugen - ein Verhalten, das sich auch in den traditionellen Programmiersprachen darstellt.

Abgesehen von den unbestreitbaren Vorteilen einer „globalen Programmiersprache“ gibt es zur Zeit keinen Grund, sich auf eine einzige Programmiersprache zu fixieren. Die Spezialisierungen der einzelnen Sprachen erfüllen Sonderaufgaben mehr oder weniger erfolgreich, die Ergonomie der Programmierung ist stark unterschiedlich, wie auch die Anforderungen der Anwender an das Resultat. Das Internet selbst erhebt Mindestanforderungen an Programmiersprachen: die Fähigkeit unterschiedliche Formate zu verarbeiten (z.B. Grafiken), Performanz (sowohl Geschwindigkeit als auch Größe), Sicherheit, Plattformunabhängigkeit, Schutz von intellektuellem Eigentum und die grundlegende Fähigkeit, mit anderen Internetanwendungen zu interagieren. Gerade die Frage nach dem Vertrauen im Internet wird heute viel diskutiert. Diese

Punkte sind offensichtlich interdependent und unterstützen unterschiedliche Zielansprüche, was die Bestimmung einer optimalen Kombination ohne Berücksichtigung des persönlichen Präferenzmusters[27] unmöglich macht.

Wichtig in der Beurteilung der zukünftigen Entwicklung im Bereich der verteilten Objekte auf Basis des Internets ist die Entwicklung der intelligenten Firewalls. Solange das ActiveX-Modell lediglich auf Vertrauensbasis operiert, können solche Komponenten nicht ohne Kontrolle in sensitiven Bereichen eines Unternehmens eingesetzt werden (vgl. Kapitel 2.4.2.5). Selbst wenn der Autor ein Mitarbeiter des Unternehmens selbst ist, ist ein unkontrollierter Einsatz einer ActiveX-Komponente nicht zu verantworten. Der konträre Sicherheitsansatz von JAVA hat zwar nicht das Problem der immanenten Unsicherheit, Applets können aber aufgrund der auferlegten Sicherheitsrestriktionen zur Zeit praktisch nicht effektiv eingesetzt werden. Hier können sichere und erprobte intelligente Firewalls eine sichere Integration von solchen verteilten Objekten in eine Intranettopologie leisten. Ohne einen solchen filternden Mechanismus zwischen Intra- und Internet, wird die globale Wiederverwendbarkeit von Objekten in absehbarer Zeit nicht erreichbar sein.

Mit dieser Entwicklung einher geht die Entwicklung der VPN. So veröffentlichte die Firma Microsoft im 3. Quartal '97 einen kostenlosen VPN Zusatz für den RAS Klienten auf dem Win95 Betriebssystem. Ohne Konfigurationsaufwand auf dem Klienten ist damit eine sichere Kommunikation mit dem RAS Server von WinNT ab Version 4.0 möglich. Die veröffentlichte Version, die sich nicht mehr in der Betaphase befindet, muß zwar noch auf Funktionsfähigkeit und Performanz untersucht werden, zeigt aber eine einfache Softwarelösung für das Intranet auf.

Eine weitere interessante zukünftige Entwicklung ist von der Firma Netscape mit ihrem „Internet Service Broker“ angekündigt. Der „Netscape Internet Service Broker (NISB)“ ist ein CORBA-basierter ORB, der im „Netscape Enterprise Server 3.0“ und auch in „Netscape Communicator 4.0“ enthalten sein soll. Anwendungsprogramme, die auf dem Server oder dem Desktop ausgeführt werden, können sich selbst mit NISB registrieren und später von anfragenden Klienten aufgerufen werden. Internetanwendungen können in JAVA, C++ oder C geschrieben werden und ihre Dienste über Internet Service Broker anbieten. Diese Erweiterungen können innerhalb des Serverprozesses oder als separater Prozeß laufen. Netscape hat eine Lizenz für die VisiBroker ORB-Technologie von Visigenics erworben und wird sie in ihre Produkte integrieren.

Ein Konsortium von Firmen[28] hat weiter eine Kooperation bei der Integration von CORBA und JAVA Beans [SUN397] vereinbart. Die Firmen haben ein gemeinsames Positionspapier veröffentlicht, welches spezifische Erweiterungen zum CORBA-Standard vorschlägt und ein Komponentenmodell etabliert, welches auch andere Modelle (wie JAVA Beans) unterstützt. Dadurch soll, mittels grafischen Entwicklungswerkzeugen, die automatische Erstellung von JAVA Beans für CORBA-Objekte ermöglicht werden [NET97].

Zum Januar 1997 wurden 12.1 Millionen Anwender des Netscape Navigator gegenüber 5.9 Millionen Anwender des MS InternetExplorer gezählt. Obwohl in Prognosen wie [CHEN97] davon ausgegangen wird, daß im Zeitraum von acht Monaten ein Gleichstand in der Anwenderzahl erreicht sein wird, bieten die Aktivitäten der Firma Netscape im Bereich des NISB eine echte Konkurrenz zu dem auf COM basierenden, ActiveX-fähigen Browser. Entscheidend dabei ist, daß die CORBA-Plattform, durch die Implementierung des NISB in dem am häufigsten verwendete WWW-Browser, eine ganz neue Bedeutung erhalten wird. Ein Kritikpunkt, nämlich die mangelnde praktische Verfügbarkeit der Plattform, könnte hierdurch in der Zukunft entkräftet werden. Damit sind auch Prognosen [CHEN97], in denen sich Microsoft mit seinem

[27] Präferenzverhalten von Personen kann die Nutzenindifferenzkurve für Gegenüberstellung von Risiko und Flexibilität stark verschieben. Einflußfaktoren können vor allem die persönliche Einschätzung des potentiellen Risikos und der damit verbunden Folgen, aber auch die Bewertung der Vorteile, die durch Akzeptanz des Risikos resultieren, sein [COL89].

[28] IBM, Netscape Communications Corporation, Oracle und Sun Microsystems.

InternetExplorer am Ende des nächsten Jahres als Monopolist etabliert hat, zumindest zu bezweifeln.

Zuletzt wird die Entwicklung der Datenbankschnittstellen ein interessanter Aspekt bleiben. Auf der Seite von JAVA muß sich die JDBC-Schnittstelle noch durchsetzten. Obwohl einige Datenbankbetreiber unter der Führung von ORACLE entsprechende Treiber angekündigt haben, steht und fällt der Erfolg von JAVA mit deren Realisierung. Im Umfeld von MS Visual Basic und COM verspricht die Einführung von ODBCDirect [DOB97] die Vorteile des RDO zu realisieren, ohne daß eine Änderung an bestehenden, auf DAO basierenden Anwendungen erforderlich wird[29]. Dies wird ein entscheidender Vorteil gegenüber dem RDO sein. Ohne Einbußen bei der Leistungsfähigkeit in der Schnittstelle zwischen Anwendung und Datenbank, bekommt so der Ruf nach Wiederverwendbarkeit, auch für bestehende OLE-Komponenten, eine neue Bedeutung. Auf der „Professional Developers Conference 1997“ in San Diego, hat Microsoft die Erweiterung COM+ angekündigt. COM+ basiert auf den integrierten Diensten und Merkmalen von COM und erleichtert Entwicklern die Erstellung und Verwendung von Softwarekomponenten unterschiedlicher Programmiersprachen und mit unterschiedlichen Werkzeugen. COM+ wird mit den MS Windows Betriebssystemen vertrieben werden und zu COM abwärtskompatibel sein (entsprechend der, in COM festgelegten, Unveränderlichkeit von Binärobjekten, vgl. Abb. 24). Zusätzlich werden Dienste angeboten werden, die zum Beispiel die Anbindung und Programmierung von Datenbankklienten vereinfachen werden [MS972] und sich in zwei zentrale Konzepte (*Datenintegration* für den direkten und einfachen Datenbankzugriff, und *Interception* um die Bindungen von Komponenten an Dienste zur Laufzeit dynamisch zu ändern) einteilen lassen [BEI97].

Diese Ankündigung zeigt das Dilemma in der Auswahl der Middleware deutlich auf. Microsoft prescht in der Entwicklung seines Komponentenmodells voran, bietet eine Vielzahl von verlockenden und anwendbaren Möglichkeiten und Diensten und eröffnet mit der Integration der Middleware in Hochsprachen einen sehr einfachen Einstieg in die Verteilung von Objekten. Ein fader Beigeschmack von Plattform- und Herstellerabhängigkeit bleibt [BRU96], zumal, in der Entwicklung von OLE zu COM, schon der Nachfolger DNA angekündigt ist [BEI97]. Die OMG bietet mit ihrem CORBA 2.0 Standard eine Alternative. Der Standard ist erprobt, offen und von vielen Herstellern anerkannt. Die praktische Anwendbarkeit, besonders in kleinen Projekten, ist aber noch unbefriedigend. Trotzdem scheint die Implementierung von verteilten Objekten auf der Basis eines „echten“ offenen Standards die richtige Wahl für zukunftssichere Entwicklungen über dem Internet als Kommunikationsplattform für Verteilte Objekte zu sein.

[29] ODBCDirect wird mit MS Office97, MS Visual Basic 5.0 und MS Visual C++ 4.2 ab Q3 97 vertrieben [DOB97].

Anhang A: Interface IUnknown

Über das IUnknown Interface können Klienten, durch die QueryInterface Methode, Zeiger auf andere Interfaces eines Objektes erhalten und die Lebenszeit des Objektes durch die Interfaces `IUnknown::AddRef` und `IUnkonwn::Release` steuern. Alle anderen COM-Interfaces sind entweder direkt oder indirekt von IUnknown abgeleitet. Damit sind die drei Methoden von IUnknown die ersten Einträge der VTable jedes Interfaces [MS941].

IUnknown muß als Teil jedes Interface implementiert werden. Wenn die verwendete Programmiersprache (z.B. C++) Mehrfachvererbung unterstützt, können verschiedene Interfaces eine Implementierung von IUnknown teilen. Falls verschachtelte Klassen verwendet werden, muß IUnknown für jedes Interface separat implementiert werden.

Die Methoden von IUnknown werden zum Wechseln zwischen den einzelnen Interfaces, innerhalb eines Komponentenobjekts, und zum Hinzufügen von Referenzen, und zum Freigeben eines Komponentenobjektes verwendet.

Die einzelnen Methoden in der Reihenfolge ihre Einträge in die VTable:

- QueryInterface   Gibt einen Zeiger auf das unterstützte Interface zurück.

```
HRESULT QueryInterface(
  REFIID iid,        //Bezeichner des angeforderten Interface
  void   **ppvObject //Adresse der Ausgabevariablen, die den
                     //Interfacezeiger enthält
);
```

Parameters
iid   [in]   Bezeichner des angeforderten Interfaces
ppvObject   [out]   Zeiger auf den angeforderten Interfacezeiger oder NIL

Return Value: S_OK, falls das Interface unterstützt wird, E_NOINTERFACE sonst

- AddRef   Inkrementiert den Referenzzähler zur Überwachung der Lebenszeit

```
ULONG AddRef(void);
```

Return Value: Gibt den aktuellen Wert des Referenzzählers als Integer von 1..n zurück

- Release   Dekrementiert den Referenzzähler

```
ULONG Release(void);
```

Return Value: Gibt den aktuellen Wert des Referenzzählers als Integer von 1..n zurück

Gerade mit der Aufgabe der Überwachung der Lebenszeit von COM-Komponenten ist dieses Interface wohl eines der wichtigsten im COM Modell insgesamt.

Anhang B: Secure Socket Layer und Private Communication Technology

„Secure Socket Layer (SSL) und „Private Communication Technology (PCT)“ sind Sicherungsprotokolle, die basierend auf einen öffentlichen Schlüssel, sichere Kanäle anbieten. Beide Protokolle werden heute bei Internet-Browsern und -Servern für die gegenseitige Authentifizierung, Nachrichtenintegrität und Datensicherheit eingesetzt. Authentifizierung von Internetservern erfolgt durch den Klienten, wenn das Zertifikat des Servers als Teil des initialen SSL/PCT Handshake präsentiert wird. Der Klient akzeptiert das Zertifikat des Servers durch Überprüfung der verschlüsselten Signatur im Zertifikat, durch ein intermediäres Zertifikat, das von einer „Certification Authority (CA)“ ausgestellt wurde oder durch einige wenige Wurzel CAs.

Authentifizierung des Klienten wird durch SSL V3.0 und PCT V1.0 unterstützt. Die folgende Darstellung verdeutlicht exemplarisch die SSL V3.0 Handshake Nachrichten zwischen Klienten und Server.

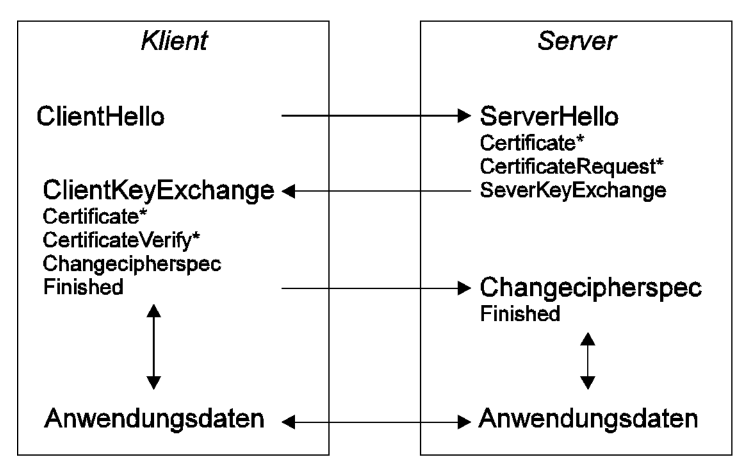


Abb. 44: SSL V3.0 Handshake.

Anhang C: Meßreihen

Für den Vergleich der Implementierungen von JAVA und ActiveX wurden zwei Versuchsaufbauten vorgenommen. Entwurfsvorgabe war vor allem die Simulation von engbandigen Übertragungskanälen zwischen Klient und Server. Um eine TCP/IP Strecke zwischen zwei Maschinen mit variablen Übertragungskapazitäten zu ermöglichen, ist eigentlich der Einsatz eines Taktgenerators erforderlich. Hierbei werden zwei Router über einen synchron getakteten Kanal verbunden. Dadurch ist eine stufenlose Regelung der Übertragungskapazität möglich.

Da diese Lösung für die vorliegende Arbeit zu kostenintensiv ist (ca. 3500,- DM für den gesamten Aufbau), wird hier ein vereinfachter Aufbau über eine Modem/Modem Kopplung gewählt. Der Klient wählt sich mit einem Modem an den auf dem WinNT Server laufenden RAS Server ein. Durch Konfiguration des Modems auf der Seite des Klienten können so Kanäle mit verschiedenen bps-Raten aufgebaut werden. Um die physikalischen Übertragungskapazitäten der einzelnen Verbindungen vergleichbar und klassifizierbar zu machen, wurde die „effektive" Antwortgeschwindigkeit der Datenübertragung mit einem Ping gemessen. Eine FTP Übertragung hätte zwar die reale Übertragungskapazität widergespiegelt, diese ist aber von zu vielen Faktoren, wie zum Beispiel Art und Qualität des Modems, Maschinenauslastung und Hardware abhängig.

Ping ist ein Instrument zur Analyse von TCP/IP Netzwerken. Wie im [RFC 1574] spezifiziert, müssen Ping-Werkzeuge die RTT für jedes übertragene Paket und die durchschnittliche minimale und maximale RTT über mehrere Pakete hinweg liefern. Ping-Pakete selbst sind dabei sehr klein, und müssen somit auch bei der Übertragung über mehrere Router nicht aufgeteilt werden. Mit der RTT können also auch topologisch stark unterschiedliche Netzwerke verglichen werden.

Als Hardwareplattformen wurden auf der Klientenseite ein Personalcomputer mit CYRIX +166 Prozessor und 64MB RAM und auf der Serverseite ein INTEL Pentium 100 mit ebenfalls 64MB RAM eingesetzt. Diese Konfiguration stellte sicher, daß die Ausführungen der Testanwendung ohne Verwendung von Auslagerungsdateien erfolgen konnte, was die Ergebnisse verfälscht hätte.



Abb. 45: Versuchsaufbau für Performanzvergleich der Middleware.

Dieser Versuchsaufbau hat gegenüber dem Taktgenerator den Nachteil, das keine stufenlose Variation des Kanals möglich ist. Nach der Hayes Spezifikation von Modems stehen nur vorgegebene Übertragungskapazitäten zu Verfügung, für die jeweils im folgenden Messungen durchgeführt wurden. Zusätzlich wurden Werte für die Interprozeßkommunikation, also die Kommunikation von Klienten- und Serverprozeß auf einer Maschine, über 10Mbit/s Ethernet LAN und einem ISDN Basiskanal ermittelt.

Das Serverobjekt wurde in diesem Versuch möglichst schmal implementiert. Es stellt nur drei Operationen (Inkrementierung, Zuweisung und eine einfache mathematische Berechnung mit allen Grundrechenarten) zur Verfügung. Die Auswahl dieser drei Grundoperationen vermeidet, das mögliche Stärken einer Programmiersprache auf einem Teilbereich das Ergebnis verfälschen.

			COM				***JavaIDL***			
Übertragungs-technik	***Effektive Datenüber-tragungs-rate [KB/s]***	***Ping [ms]***	***Ver-bindungs-aufbau***	***Zu-weisung***	***Inkremen-tierung***	***Opera-tionen***	***Ver-bindungs-aufbau***	***Zu-weisung***	***Inkremen-tierung***	***Opera-tionen***
			1710	2,73	3,40	4,07	710	7,58	7,31	8,79
			1010	3,87	2,54	3,26	770	7,74	7,58	8,40
			590	3,23	3,79	2,40	770	7,75	7,47	8,62
			950	3,95	4,48	3,48	770	7,63	7,25	8,41
			1620	2,37	2,99	3,55	660	7,64	7,52	8,41
			2130	2,77	3,33	3,94	710	7,69	7,42	8,46
			1540	3,17	2,73	3,46	720	7,69	7,96	8,52
			1640	2,27	2,83	3,50	710	8,13	7,42	8,73
			780	3,41	2,97	2,64	550	7,96	8,13	8,89
			1630	2,33	2,89	3,50	770	7,80	7,41	8,68
Interprozeß			**1360**	**3,01**	**3,20**	**3,38**	**714**	**7,76**	**7,55**	**8,59**
			990	7,25	6,07	7,22	22910	6,97	6,81	7,80
			1380	5,58	6,45	5,66	23340	7,03	6,87	7,80
			1290	5,50	6,32	7,03	23130	7,03	6,92	7,86
			1280	6,04	6,80	5,52	23230	6,97	6,87	7,91
			2190	6,83	6,25	7,07	23180	7,08	6,87	7,80
			2680	6,83	5,60	6,31	23950	7,03	6,81	7,85
			1500	5,65	6,47	7,23	23780	7,03	6,87	7,79
			2530	7,12	6,44	7,21	22850	7,03	6,86	7,80
			1360	5,45	6,22	6,99	22690	7,03	6,92	7,80
			1350	5,50	6,32	7,09	22690	7,03	6,86	7,86
LAN (Ethernet)	546,8	1	**1655**	**6,18**	**6,29**	**6,73**	**23175**	**7,02**	**6,87**	**7,83**
			14690	1585,50	1586,40	1588,80	26200	617,40	617,40	638,20
			15620	1573,20	1547,50	1587,30	25160	617,40	625,60	686,50
			14800	1592,90	1586,70	1570,60	25430	612,40	622,30	680,50
			14720	1558,90	1589,60	1540,50	24770	611,30	621,20	690,40
			14280	1575,10	1590,30	1442,80	26580	614,70	617,90	689,30
			18390	1571,10	1543,70	1556,00	26090	623,40	617,90	689,90
			15740	1573,30	1577,80	1564,70	25930	615,10	652,20	689,40
			13990	1560,00	1555,10	1591,90	26640	617,30	617,90	688,80
			15030	1579,60	1568,70	1534,90	25430	617,90	613,50	882,80
			16150	1574,20	1564,40	1548,70	27020	628,90	626,70	682,70
Modem 2400	1,9	928	**15341**	**1574,38**	**1571,02**	**1552,62**	**25925**	**617,58**	**623,26**	**701,85**
			8430	778,80	777,50	752,60	25050	351,00	351,50	377,40
			8400	767,40	772,40	731,10	24940	352,10	350,50	378,40
			8090	762,60	764,80	742,80	24160	351,60	351,50	378,40
			8130	764,10	765,80	743,30	24990	352,10	350,90	377,90
			8340	765,60	766,60	742,00	24500	351,00	350,90	376,80
			7400	753,40	752,30	727,20	24720	352,00	351,00	379,50
			8470	774,70	773,40	749,50	24660	351,50	351,00	379,00
			8300	765,20	767,90	747,00	24880	351,60	350,90	377,40
			8160	762,70	762,10	738,50	24770	351,50	351,50	377,90
			7370	752,70	752,20	728,60	25600	351,50	350,40	376,80
Modem 4800	0,5	457	**8109**	**764,72**	**765,50**	**740,26**	**24827**	**351,59**	**351,01**	**377,95**

			5460	340,10	345,90	363,90	24010	200,64	200,59	213,11
			4320	354,30	361,20	370,80	24170	200,53	200,59	213,33
			5130	362,30	370,90	351,10	23230	200,48	200,48	213,17
			4960	365,00	373,00	361,00	23290	200,42	200,37	213,94
			5320	367,60	352,80	361,40	24010	200,47	200,43	212,56
			5880	252,70	359,50	366,40	24830	200,64	200,42	213,17
			4820	365,90	372,20	360,70	24880	200,47	200,54	213,44
			4620	360,60	365,30	372,70	25050	200,64	200,37	213,61
			5790	373,30	358,60	367,10	24230	200,53	200,48	212,94
			6160	357,60	364,40	371,90	24380	200,48	200,54	212,94
Modem 9600	1,0	264	**5246**	**349,94**	**362,38**	**364,70**	**24208**	**200,53**	**200,48**	**213,22**
			8540	234,50	239,90	253,20	23940	143,19	143,08	152,03
			3550	246,50	249,70	232,70	23840	143,24	143,08	152,20
			3820	226,50	238,50	250,00	24000	143,19	143,03	152,80
			3650	245,50	237,50	259,00	23240	143,57	143,03	151,81
			2890	238,80	233,10	245,10	23230	143,25	142,86	152,30
			3010	242,20	234,20	225,20	23950	143,85	142,97	153,08
			4670	240,50	232,60	244,10	23400	143,41	143,19	152,25
			3950	232,70	244,80	236,80	23450	143,52	143,09	152,97
			3250	227,90	240,50	233,10	24450	143,52	143,13	152,42
			3050	243,10	236,30	240,00	23780	143,30	143,08	152,53
Modem 19200	1,9	189	**4038**	**237,82**	**238,71**	**241,92**	**23728**	**143,40**	**143,05**	**152,44**
			2400	169,70	157,70	174,10	24220	120,50	120,57	126,93
			2520	163,10	163,90	161,90	23680	120,56	120,67	126,82
			2990	158,05	158,78	155,58	24280	120,56	120,61	126,55
			2110	161,20	160,30	176,70	23120	121,05	120,95	127,48
			3400	164,10	171,00	165,70	23010	120,94	120,32	126,32
			3260	158,94	157,06	165,26	23120	120,39	120,97	124,70
			3400	174,10	171,60	176,90	23400	120,83	120,95	125,80
			2090	161,10	158,60	174,40	23730	120,81	120,31	124,73
			2410	165,40	165,60	174,10	23290	120,84	120,89	127,26
			3590	158,05	158,06	156,59	22950	120,95	121,00	127,37
Modem 33600	3,4	122	**2817**	**163,37**	**162,26**	**168,12**	**23480**	**120,74**	**120,72**	**126,40**
			2670	45,88	44,31	44,68	22570	40,05	40,07	40,15
			1630	44,89	45,32	45,64	22680	40,10	40,04	40,09
			1930	45,14	45,57	45,94	23340	40,10	40,04	40,04
			1140	44,4	44,94	45,32	23180	40,04	40,04	40,04
			1520	44,83	45,26	45,53	22570	40,04	40,04	40,10
			1200	44,4	45,49	44,52	23400	40,10	40,09	40,09
			2010	45,27	45,7	46,02	23290	40,04	40,04	40,10
			1890	45,15	45,58	45,9	23290	40,10	40,04	40,04
			1560	44,76	45,08	45,29	23570	40,04	40,04	40,09
			2270	45,58	45,96	45,28	23230	40,04	40,10	40,08
ISDN 64000	7,9	27	**1782**	**45,03**	**45,32**	**45,41**	**23112**	**40,07**	**40,05**	**40,08**

Tab. 25: Performanzvergleich zwischen COM und JavaIDL.

Für die Messung der Leistungsfähigkeit der Komponenten bei der Datenanbindung wurde ein analoger Versuchsaufbau gewählt.



Abb. 46: Versuchsaufbau für Performanzvergleich von RDO und JDBC-ODBC.

Für beide Systeme wurde sowohl die Zeit, die für die Instantiierung der eigentlichen Verbindung benötigt wurde, als auch die Zeit für die Erstellung des lokalen Recordsets gemessen. Ein Vergleich der Transaktionszeiten selbst ist nicht erforderlich, da hier die zugrundeliegenden Mechanismen der ODBC-API von beiden Systemen gleich verwendet werden, ohne das ein weiterer Overhead von einem der Systeme implementiert wird.

	RDO		***JDBC-ODBC***	
Übertragungs-technik	***Verbindungs-aufbau [ms]***	***Recordset erstellen [ms]***	***Verbindungs-aufbau [ms]***	***Recordset erstellen [ms]***
	870	890	1490	160
	1010	520	1100	170
	1070	250	1040	160
	1630	520	1040	170
	1300	780	1050	160
	1640	290	990	170
	1540	340	1050	220
	1900	500	1040	170
	1270	930	1050	160
	1630	240	1040	170
LAN (Ethernet)	**1443**	**486**	**1044**	**172**

Tab. 26: Performanzvergleich von RDO und JDBC-ODBC Brücke.

Anhang D: Funktionsreferenz für DCE und MS RPC-API Funktionen

MS RPC V2.0 unterstützt nahezu alle OSF DCE RPC-API und bietet selbst einige Erweiterungen. Die Beziehungen zwischen den beiden Bezeichnungsschemata werden in der folgenden Tabelle dargestellt. Besonders bemerkenswert sind die Veränderungen seit der Veröffentlichung der MS RPC V1.0 [ROS932]. Inkompatibilitäten zwischen den Implementierungen werden hervorgehoben.

DCE Funktionsname	*MS RPC Funktionsname*
dce_error_inq_text	DceErrorInqTest
nicht unterstützt	RpcAbnormalTermination[1)3)]
rpc_binding_copy	RpcBindingCopy[3)]
rpc_binding_free	RpcBindingFree[3)]
rpc_bindin_from_string_binding	RpcBindingFromStringBinding[3)]
nicht unterstützt	RpcEndExcept[1)]
nicht unterstützt	RpcEndFinally[1)]
nicht unterstützt	RpcExcept[1)]
nicht unterstützt	RpcExceptionCode[1)]
nicht unterstützt	RpcFinally[1) 3)]
rpc_binding_inq_auth_dient	RpcBindingInqAuthClient[3)]
rpc_binding_inq_auth_info	RpcBindinglnqAuthlnfo[3)]
rpc_binding_inq_object	RpcBindinglnqObject[3)]
rpc_binding_reset	RpcBindingReset[3)]
rpc_binding_server_from_client	RpcBindingServerFromClient[3)]
rpc_binding_set_auth_info	RpcBindingSetAuthInfo[3)]
rpc_binding_set_object	RpcBindingSetObject[3)]
rpc_binding_to_string_binding	RpcBindingToStringBinding[3)]
rpc_binding_vector_free	RpcBindingVectorFree[3)]
rpc_ep_register	RpcEpRegister[3)]
rpc_ep_register_no_replace	RpcEpRegisterNoReplace[3)]
rpq_ep_resolve_binding	RpcEpResolveBinding[3)]
rpc_ep_unregister	RpcEpUnregister[3)]
rpc_if_id_vector_free	RpcIfIdVectorFree[3)]
rpc_if_inq_id	RpcIfInqId[3)]
rpc_if_register_auth_info	**nicht unterstützt**[3)]
nicht unterstützt	RpclmpersonateClient[3)]
nicht unterstützt	RpcMgmtEnableIdleCleanup[3)]
rpc_mgmt_ep_elt_inq_begin	RpcMgmgEpEltInqBegin[3)]
rpc_mgmt_ep_elt_inq_done	RpcMgmgEpEltInqDone[3)]
rpc_mgmt_ep_elt_inq_next	RpcMgmgEpEltInqNext[3)]
rpc_mgmt_ep_unregister	RpcMgmgEpEltInqBegin[3)]
rpc_mgmt_inq_com_timeout	RpcMgmtInqComTimeout[3)]
rpc_mgmt_inq_dflt_authn_level	**nicht unterstützt**[3)]
rpc_mgmt_inq_dflt_protect_level	RpcMgmtInqDefaultProtectLevel
rpc_mgmt_inq_if_ids	RpcMgmtInqIfIds[3)]
rpc_mgmt_inq_server_princ_name	RpcMgmtInqServerPrincName[3)]
rpc_mgmt_inq_stats	RpcMgmtInqStats[3)]
rpc_mgmt_is_server_listening	RpcMgmtIsServerListening[3)]

DCE Funktionsname	***MS RPC Funktionsname***
rpc_mgmt_set_authorization_fn	RpcMgmtSetAuthorizationFn[3]
rpc_mgmt_set_cancel_timeout	RpcMgmtSetCancelTimeout[3]
rpc_mgmt_set_com_timeout	RpcMgmtSetComTimeout[3]
rpc_mgmt_set_server_stack_size	RpcMgmtSetServerStackSize[3]
rpc_mgmt_stats_vector_free	RpcMgmtStatsVectorFree[3]
rpc_mgmt_stop_server_listening	RpcMgmtStopServerListening[3]
nicht unterstützt	RpcMgmtWaitServerListen[3]
rpc_network_inq_protseqs	RpcNetworkInqProtseqs[3]
rpc_network_is_protseq_valid	RpcNetworkIsProtseqValid[3]
rpc_ns_binding_export	RpcNsBindingExport[3]
rpc_ns_binding_import_begin	RpcNsBindinglmportBegin[3]
rpc_ns_binding_import_done	RpcNsBindinglmportDone[3]
rpc_ns_binding_import_next	RpcNsBindinglmportNext[3]
rpc_ns_binding_inq_entry_name	RpcNsBindinglnqEntryName
rpc_ns_binding_lookup_begin	RpcNsBindingLookupBegin[3]
rpc_ns_binding_lookup_done	RpcNsBindingLookupDone[3]
rpc_ns_binding_lookup_next	RpcNsBindingLookupNext[3]
rpc_ns_binding_select	RpcNsBindingSelect[3]
rpc_ns_binding_unexport	RpcNsBindingUnexport[3]
rpc_ns_entry_expand_name	RpcNsEntryExpandName[3]
rpc_ns_entry_object_inq_begin	RpcNsEntryObjectInqBegin[3]
rpc_ns_entry_object_inq_done	RpcNsEntryObjectInqDone[3]
rpc_ns_entry_object_inq_next	RpcNsEntryObjectInqNext[3]
rpc_ns_group_delete	RpcNsGroupDelete[2] [3]
rpc_ns_group_mbr_add	RpcNsGroupMbrAdd[2] [3]
rpc_ns_group_mbr_inq_begin	RpcNsGroupMbrInqBegin[2] [3]
rpc_ns_group_mbr_inq_done	RpcNsGroupMbrInqDone[2] [3]
rpc_ns_group_mbr_inq_next	RpcNsGroupMbrInqNext[2] [3]
rpc_ns_group_mbr_remove	RpcNsGroupMbrRemove[2] [3]
ipc_ns_mgmt_binding_unexport	RpcNsMgmtBindingUnexport[3]
rpc_ns_mgmt_entry_create	RpcNsMgmtEntryCreate[3]
rpc_ns_mgmt_entry_delete	RpcNsMgmtEntryDelete[3]
rpc_ns_mgmt_entry_inq_if_ids	RpcNsMgmtEntryInqIfIds[3]
rpc_ns_mgmt_handle_set_exp_age	RpcNsMgmtHandleSetExpAge[3]
rpc_ns_mgmt_inq_exp_age	RpcNsMgmtInqExpAge[3]
rpc_ns_mgmt_set_exp_age	RpcNsMgmtSetExpAge
rpc_ns_profile_delete	RpcNsProfileDelete[3]
rpc_ns_profile_elt_add	RpcNsProfileEltAdd[3]
rpc_ns_profile_elt_inq_begin	RpcNsProfileEltInqBegin[3]
rpc_ns_profile_elt_inq_done	RpcNsProfileEltInqDone[3]
rpc_ns_profile_elt_inq_next	RpcNsProfileEltInqNext[3]
rpc_ns_profile_elt_remove	RpcNsProfileEltRemove[3]
rpc_object_inq_type	RpcObjectInqType[3]
rpc_object_set_inq_fn	RpcObjectsetInqFn[3]
rpc_object_set_type	RpcObjectSetType[3]
rpc_protseq_vector_free	RpcProtseqVectorFree[3]

DCE Funktionsname	*MS RPC Funktionsname*
nicht unterstützt	RpcRaiseException[3)]
nicht unterstützt	RpcRevertToSelf[3)]
rpc_server_inq_bindings	RpcServerInqBindings[3)]
rpc_server_inq_if	RpcServerInqIf[3)]
rpc_server_listen	RpcServerListen[3)]
rpc_server_register_auth_info	RpcServerRegisterAuthInfo(no DLL) [3)]
rpc_server_register_if	RpcServerRegisterIf[3)]
rpc_server_unregister_if	RpcServerUnregisterIf[3)]
rpc_server_use_all_protseqs	RpcServerUseAllProtseqs[3)]
rpc_server_use_all_protseqs_if	RpcServerUseAllProtseqsIf[3)]
rpc_server_use_protseq	RpcServerUseProtseq[3)]
rpc_server_use_protseq_ep	RpcServerUseProtseqEp[3)]
rpc_server_use_protseq_if	RpcServerUseProtseqIf[3)]
rpc_ss_allocate	RpcSsAllocate
rpc_ss_destroy_client_context	RpcSsDestroyClientContext[3)]
rpc_ss_disable_allocate	RpcSsDisableAllocate
nicht unterstützt	RpcSsDontSerializeContext
rpc_ss_enable_allocate	RpcSsEnableAllocate
rpc_ss_free	RpcSsFree
rpc_ss_get_thread_handle	RpcSsGetThreadHandle
rpc_ss_register_auth_info	**nicht unterstützt**
rpc_ss_set_client_alloc_free	RpcSsSetClientAllocFree
rpc_ss_set_thread_handle	RpcSsSetThreadHandle
rpc_ss_swap_client_alloc_free	RpcSsSwapClientAllocFree
rpc_string_binding_compose	RpcStringBindingCompose[3)]
rpc_string_binding_parse	RpcStringBindingParse[3)]
rpc_string_free	RpcStringFree[3)]
nicht unterstützt	RpcTestCancel[3)]
nicht unterstützt	RpcTryExcept[1)]
nicht unterstützt	RpcTryFinally[1)]
nicht unterstützt	RpcWinSetYieldInfo[3)]
nicth unterstützt	RpcWinSetYieldTimeout[3)]
uuid_compare	UuidCompare[3)]
uuid_create	UuidCreate[3)]
uuid_crate_nil	UuidCreateNil[3)]
uuid_equal	UuidEqual[3)]
uuid_from_string	UuidFromString[3)]
uuid_hash	UuidHash[3)]
uuid_is_nil	UuidIsNil
uuid_to_string	UuidToString[3)]
nicht unterstützt	YieldFunctionName[3)]

Tab. 27: Funktionsreferenz für DCE und MS RPC-API V2.0 Funktionen.

[1)] MS Makro für Behandlung von Ausnahmefehlern

[2)] Wird nur durch CDS unterstützt

[3)] Wird nicht unter Macintosh unterstützt (zum Teil, weil DCOM noch nicht auf Macintosh-Betriebssystemen MacOS verfügbar ist).

Anhang E: Unterschiede in den IDL von OSF DCE und MS DCE

Die beiden IDL verwenden ähnliche Datenbasistypen. In der folgenden Tabelle werden die einzelnen Typen erläutert und Unterschiede dargestellt [ROS932].

IDL Typen	***MIDL Typen***	***Erläuterung***
Datentypen		
boolean	boolean	
byte	byte	
void	void	
void*	void*	
handle_t oder rpc_binding_handle	handle_t oder RPC_HANDLE	Beide Bezeichner sind jeweils äquivalent. Die handle_t Bezeichner werden nur wegen Abwärtskompatibilität gehalten.
error_status_t	error_status_t	DCE konvertiert error_status_t zu dem host-eigenen Fehler-statusformat. WinNT konvertiert error_status_t Argumente anhand von winerror.h.
Integer		
small	small	
short	short	
long	long / int	
hyper	hyper	wird nach MIDL V1.0 unterstützt
unsigned small	unsigned small	
unsigned short	unsigned short	
unsigned long	unsigned long	
unsigned hyper		Keine Unterstützung unter MIDL
	wchar_t	Keine Unterstützung unter IDL
Floating Point		
float	float	
double	double	
Internation Charachters		
ISO_LATIN_1 ISO_UCS ISO_MULTI_LINGUAL		Obwohl MIDL diese Charachtersets nicht unterstützt, kann man über wchar_t UNICODE emulieren.[1)]

Tab. 28: Unterschiede in der IDL bei OSF DCE und MS DCE.

[1)] Die aktuelle Version 4.0 Win32-SDK von Microsoft hat an dieser Stelle einen Fehler. Die Typendefinition des TCHAR_T sollte aus der Datei WTYPES.IDL entfernt werden um nicht mit WCHAR_T verwechselt zu werden, auch wenn UNICODE nicht verwendet wird [MSKB973].

Anhang F: Inhalt der Disketten im Anhang

Im folgenden wird der Inhalt der beigefügten Disketten kurz aufgeführt, und die Funktionen der wichtigsten Dateien erläutert.

Diskette 1: JavaIDL Implementierung und Benchmarks:

- Browser
  - Makefile
  - Classes: Verzeichnis der Bytecode-Klassen.
  - Dokumentation
    - jdbc.RemoteRecordSet.html: Dokumentation der RemoteRecordSet-Klasse
    - Images
  - IDL: IDL Spezifikationsdateien
    - IDL
      - Benchmark
        - Benchmark.idl: IDL der JDBC Benchmark-Anwendung
      - Browser
        - Browser.idl: IDL des Datenbankservers
        - Ping.idl: IDL des Datenbankklienten
  - JAVA
    - IDL
      - Benchmark
        - BenchClient.java: JAVA-Benchmark-Klient
        - BenchImpl.java: JAVA-Serverobjektimplementierung
        - BenchServer.java: JAVA-Benchmark-Server
      - Browser
        - BrowserClient.java: JAVA-Datenbankklient mit GUI
        - BrowserImpl.java: JAVA-Serverobjektimplementierung
        - BrowserServer.java: JAVA-Datenbankserver
        - PingImpl.java: JAVA-Implementierung der Trigger-Methode
    - JDBC
  - Pages
    - browser.html: HTML-Seite des Datenbankklienten
    - benchmark.html: HTML-Seite des Klienten für den Benchmarkversuch
  - Stubs
    - IDL
      - Benchmark: Stubs des Benchmarkversuches
      - Browser: Stubs des Datenbankbeispiels

Diskette 2: DCOM Benchmarks:

🗁 DCOM Bench	Benchmarkversuch für die COM Middleware
🗁 Client	
🗎 modDCOMBenchClient.bas	Implementierung des Klienten
🗎 DCOMBenchClient.vbp	Visual Basic Projektdatei
🗎 DCOMBenchClient.exe	Ausführbarer Klient für den Benchmarkversuch
🗁 Server	
🗎 clsDCOMBenchServer.cls	Implementierung der öffentlichen Serverklasse
🗎 DCOMBenchServer.exe	Ausführbarer Server für den Benchmarkversuch
🗎 DCOMBenchServer.vbp	MS Visual Basic Projektdatei
🗁 RDO Data Bench	Benchmarkversuch für den Vergleich von DAO und RDO
🗁 Client	
🗎 DCOMDataClient.exe	Ausführbarer Klient für den Benchmarkversuch
🗎 DCOMDataClient.vbp	MS Visual Basic Projektdatei
🗎 MainFrame.frm	GUI des Klienten
🗎 MainModule.bas	Unterstützendes Modul für die GUI
🗁 Server	
🗎 DataServer.cls	Öffentliche Serverklasse für den Benchmarkversuch
🗎 DCOMDataServer.exe	Ausführbarer Server für Benchmarkversuch
🗎 DCOMDataServer.vbp	Visual Basic Projektdatei

Diskette 3: TriggerHelper:

🗎 Hello.mak	MS C++ Projektdatei für „TriggerHelper“
🗎 NativeMethodShell.C	Generierter Serverstub für native JAVA Aufrufe
🗎 NativeMethodShellIMP.C	Implementierung des Stubs
🗎 TriggerHelper.dll	Kompilierte MS Windows DLL
🗎 XP.C	Implementierung der IPC Kommunikation

Literaturverzeichnis

[BEI97] Beigl, M.; Segor, C.:
COMe together, Microsofts Distributed interNet Applications Architecture,
erschienen in: iX, Jg. 12 (1997), S. 136-140.

[BER96] Berger, J.: *Die Kryptographie in der Bürokommunikation*,
Siemens Nixdorf White Paper (intern), 1996.

[BRU96] Simpson, B.; Mitchell, J.; Christeson, B.; u.a.m.:
Making Sense of Java, A Guide for Managers and the Rest of Us,
Manning Publications Co., Greenwich, 1996.

[BRU97] Bruce, G.; Dempsey, R.: *Security in Distributed Computing*,
Prentice Hall, Saddle River, 1997.

[CAM97] Campione, M.: *Integrating Native Methods into JAVA Programs*,
Sun Microsystems Inc., 8. Juli 1997.
http://java.sun.com/docs/books/tutorial/native/

[CHAP96] Chappel, D.: *Understanding ActiveX and OLE*, Microsoft Press, Redmond, 1996.

[CHEN97] Chen, H. W.; James, B.; Tsai, C. T.; u.a.m.:
Cyber-Battle: Netscape Versus Microsoft: A Dynamic Analysis, Juni 1997.
http://www-leland.stanford.edu/~johntsai/dypro.html

[CHES94] Cheswick, W. R.; Bellovin, S. M.: *Firewalls and Internet Security*,
Addison-Wesley, 1994.

[CHO96] Chorus: *Cool ORB 3.1 News Release*, Chorus Systems. 1996.
http://www.chorus.com/News/coolorb.html

[CHP97] Check Point: *Check Point FireWall-I White Paper*,
Check Point - White Paper, 1997.
http://www.checkpoint.com

[CHU97] Chung, P. E.; Huang, Y.; Yajink, S.:
DCOM and CORBA Side by Side, Step by Step, and Layer by Layer,
3. Sept. 1997.
http://www.research.att.com/~ymwang/papers/HTML/DCOMnCORBA/S.html

[COL89] Busse von Colbe, W.; Hammann, P.; Laßmann, G.: *Betriebswirtschaftstheorie II*,
3. Aufl., Springer, Heidelberg - New York - London u.a., 1989.

[COO96] Cook, D.: *Write a Simple HTTP-based Server Using MFC and Windows Sockets*,
erschienen in: Microsoft Systems Journal, Vol. 11 (1996), No. 2.

[CT1209] Schmitt, Dr. H.-J.: *Bewegliche Ziele, ActiveX: Microsofts Antwort auf JAVA*,
erschienen in: C’T, Nr. 12 (1996), S. 258-264.

[CW96] Computerwoche:
Konkurrenz zwischen CORBA und DCOM, Das Intranet wandelt sich zur Plattform für Verteilte Objekte,
erschienen in: Computerwoche, Nr. 45 (1996), S. 27-28.

[DEC94] Digital Equipment Corporation:
Digital Distributed Computing Environment (DCE) for Windows NT,
Digital Equipment Corporation, 1994.

[DIC97] Dicken, H.: *Formel SQL, Performance von Datenbankabfragen aus Java*,
erschienen in: iX, Jg. 12 (1997), S. 124-129.

[DOB97] Dobson, R.: *Go Directly to ODBC with ODBCDirect*, erschienen in:
MS Interactive Developer Nr. 11 (1997), S. 66-75.

[DOD85] Department of Defence: *The Orange Book*,
Department of Defence Standard, DoD 5200.28-STD, Dezember 1985.
http://www.disa.mil/MLS/info/orange/

[DOS96] Doster, M.: *JAVA im/und Internet*, 17. Juni 1996.
http://www.informatik.mu-luebeck.de/oster/studium/java.html

[DRAFT95] Hickman, K.; Elgamal, T.: *The SSL Protocol*,
Internet - Draft, Juni 1995.
http://ds.internic.net/internet-drafts/draft-hickman-netscape-ssl-01.txt

[DRAFT961] Brown, N.; Kindel, C.:
Distributed Component Object Model Protocol - DCOM/1.0,
Internet - Draft, 2. Mai 1996.
http://ds.internic.net/internet-drafts /draft-brown-dcom-v1-spec-00.txt

[DRAFT962] Hamzeh, K.; Singh P. G.; Verthein, W.; u.a.m.:
Point-to-Point Tunnelling Protocol—PPTP,
Internet - Draft, Juni 1996.
http://ds.internic.net/internet-drafts/draft-ietf-pppext-pptp-00.txt

[FRA96] Franklin, C.: *Visual Basic 4.0 Internet Programming*,
John Wiley & Sons Inc., Toronto, 1996.

[HAL96] Halsall, F.: *Data Communications, Computer Networks, and Open Systems*,
4. Aufl., Wokingham u.a., 1996.

[HER93] Engesser, H.: *Duden Informatik*,
2. Aufl., Duden Verlag, Mannheim - Leipzig - Wien u.a., 1993.

[HM95] Heller, M.: *Tips and Tricks on Developing Killer Server Applications for Windows NT*, erschienen in: Microsoft Systems Journal, No. 8 (1995).

[INT96] Intersolv: *Intersolv and Sun Partnership Bridges the JDBC-ODBC Gap*,
Sweet JAVA - White Paper, 1996.
http://www.intersolv.com/programs/sweet_white.htm

[IONA197] IONA Technologies: *IIOP on the Internet (Firewall Navigation and WonderWall)*
IONA Technologies White Paper, 1997.

[IONA297] IONA Technologies: *IONA Technologies Announces Integrated Suite to Provide Full Enterprise Service Infrastructure*,
IONA Technologies Press Release, 5. März 1997.
Http://www.iona.com/press/pr/otm.html

[JEP97] Jepson, B.: *JAVA Database Programming*,
John Wiley & Sons Inc., 1997.

[KES94] Gartner, H. A.; Konrad, P.: *So sehen DV-Anwender ihre Sicherheit*,
erschienen in: KES, Jg. 10 (1994), Heft 3.

[KIN95] Kindel, C.: *Designing COM Interfaces*
Veröffentlicht im Microsoft Developers Network,
Microsoft Corporation, 1995.

[KIR95] Kirk, D. van: *Diamond in the Rough*,
erschienen in: Information Week, 31. Juli 1995.

[LIN96] Linthicum, D. S.: *The JDBC Connection*,
erschienen in: Internet Systems, Oktober 1996.

[MER97] Merkle, B.: *In die Ferne schweifen, RMI: Verteilte Java-Objekte*,
erschienen in: iX, Jg. 12 (1997), S. 130-135.

[MOA96] Moazzam, A.; Gilroy, J.; Perry, D.; u.a.m.: *DNS and Microsoft Windows NT 4.0*,
PDC Conference White Paper, 1996.

[MOR97] Morrison, M.: *JAVA Unleashed*,
2. Aufl., Sams.net Publishing, 1997.

[MOV95] Mowbray, T. J.; Zahavi, R.:
The Essential CORBA, Systems Integration Using Distributed Objects,
John Wiley & Sons Inc., New York - Chichester - Brisbane u.a., 1995.

[MS941] Mircosoft: *COM Programmer's Reference*,
Veröffentlicht im Microsoft Developer Network,
Microsoft Corporation, 1994.

[MS942] Microsoft: *Microsoft Press Computer Dictionary*
Microsoft Corporation, 1994.

[MS951] Mircosoft: *Building Client/Server Applications with Visual Basic*,
Microsoft Corporation, 1995.

[MS952] Microsoft: *The Component Object Model Specification*,
Microsoft Corporation und Digital Equipment Corporation,
Draft Version 0.9, 24. Oktober, 1995.
www.microsoft.com/oledev/olecom/title.htm

[MS961] Microsoft: *Remote Server Connectivity*, Mircosoft Corporation, 1996.

[MS962] Microsoft: *Microsoft Visual Basic 4.0 Benutzerhandbuch*,
(Online Dokumentation), Mircosoft Corporation, 1996.

[MS963] Microsoft: *Microsoft Windows NT Server Resource Kit Supplement 1 book*,
(Online Dokumentation), Mircosoft Corporation, 1996.

[MS964] Microsoft:
What Is the Exploder Control and How Does It Relate to Authenticode?
Mircosoft Corporation, 3. September 1996.

[MS965] Microsoft: *DCOM Technical Overview*,
Microsoft Corporation, PDC Conference White Paper, 1996.

[MS966] Microsoft: *Microsoft Windows NT Distributed Security Services*,
Microsoft Corporation, PDC Conference White Paper, 1996.

[MS971] Microsoft: *Microsoft Visual Basic 5.0 Benutzerhandbuch*,
(Online Dokumentation), Mircosoft Corporation, 1997.

[MS972] Microsoft: *Microsoft Announces COM+*,
Microsoft Corporation, Press Release, 23. September 1997.
http://www.microsoft.com/corpinfo/press/1997/Setp97/COMplspr.htm

[MSDBLIB] Microsoft: *Programming DB-Library for Visual Basic (Online Dokumentation)*,
Microsoft Press, 1997.

[MSDN997] *Data Access with Visual J++ and ADO for JAVA*, erschienen in:
Microsoft Developer Network News Nr. 5 (1997), S. 1-8.

[MSKB971] Microsoft Knowledge Base: *Discussion of DCE CDS and RPC NSI*
Article ID: Q103738, Microsoft Corporation, 1997.

[MSKB972] Microsoft Knowledge Base:
RegisterDatabase Method does not modify ODBC.INI File,
Article ID: Q132329, Microsoft Corporation, 1997.

[MSKB973] Microsoft Knowledge Base: *BUG: Win32 SDK Version 4.0 Bug List - MIDL*,
Article ID: Q165280, Microsoft Corporation, 1997.

[MSRPC92] Microsoft: *Microsoft RPC Programmer's Guide and Reference*
(Online Dokumentation), Microsoft Press, 1992.

[MSRPC97] Microsoft: *Microsoft RPC Programmer's Guide and Reference*,
(Online Dokumentation), Microsoft Press, 1997.

[MSSDK97] Microsoft: *Microsoft Platform SDK for Microsoft Windows*, (Online Dokumentation), Microsoft Corporation, 1997.

[MSSQL96] Microsoft: *Microsoft SQL Server Setup*, (Online Dokumentation), Microsoft Press, 1996.

[NET97] Netscape: *IBM, Netscape, Oracle and SUN unite to bring CORBA and JavaBeans together*, Netscape Communications Corporation, 10. Juni 1997. http://search.netscape.com/de/newsref/pr/newsrelease422.html

[NOR96] Northon, K.: *Sun's JDBC spec turns up database heat under JAVA*, erschienen in: PC Woche, Mai 1996. http://www.pcweek.com/archieve/960506/pcwk0060.htm

[OMG95] OMG: *The Common Object Request Broker: Architecture and Specification*, Revision 2.0, Juli 1995, Aktualisiert Juli 1996.

[OMG96] OMG: *Comparing ActiveX and CORBA/IIOP*, OMG, 1996. http://www.omg.org/activex.htm

[OMG97] OMG: *JAVA Language Mapping RFP (ORBOS RFP3)*, OMG, 1997. http://www.omg.org/library/schedule/ORBOS_RFP3.htm

[ORF97] Robert O., Dan H.: *Client/Server Programming with JAVA and CORBA*, Wiley Computer Publishing, New York - Chichester - Weinheim u.a., 1997.

[OSC97] Ford, S.; Wells, D.; Wells, N.: *Internet Tool Survey*, Object Services and Consulting Inc., 1997. http://www.objs.com/survey/survey.htm

[RED96] Redlich, J.-P.: *CORBA 2.0, Praktische Einführung für C++ und JAVA*, Addison-Wesley, Bonn - Reading - Menlo Park u.a., 1996.

[RFC 1034] Mockapetris, P.: *Domain Names - Concepts and Facilities*, November 1987.

[RFC 1035] Mockapetris, P.: *Domain Names - Implementation and Specification*, November 1987.

[RFC 1508] Linn, J.: *Generic Security Service Application Program Interface*, September 1993.

[RFC 1510] Kohl, J.; Neuman, C.: *The Kerberos Network Authentication Service (V5)*, September 1993.

[RFC 1521] Borenstein, N.; Freed, F.: *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*, September 1993.

[RFC 1522] Moore, K.: *MIME (Multipurpose Internet Mail Extensions) Part Two: Message Header Extensions for Non-ASCII Text*, September 1993.

[RFC 1574] Hares, S.; Wittbrodt, C.: *Essential Tools for the OSI Internet*, Februar 1994.

[RFC 1661] Simpson, W.: *The Point-to-Point Protocol (PPP)*, Juli 1994.

[RFC 1701] Hanks, S.; Farinacci, D.; Traina, P.: *Generic Routing Encapsulation (GRE)*, Oktober 1994.

[RIC90] Ricardo, C.: „*Database Systems: Principles, Design and Implementation*, Maxwell Macmillan, New York, 1990.

[ROD97] Rodley, J.: *Developing Databases for the Web & Intranets*,
The Coriolis Group Inc., Scottsdale, 1997.

[ROM97] Roman, S.: *Concepts of Object-Oriented Programming with Visual Basic*,
Microsoft Corporation, 1997.
http://www.microsoft.com/vbasic/feature/object/objwebdoc.htm

[ROS931] Rosenberrry, W.; Kenney, D.; Fischer, G.: *Understanding DCE*,
2. Aufl., O`Reilly & Associates Inc., Sebastopol, 1993.

[ROS932] Rosenberrry, W.; Teague, J.:
Distributing Applications Across DCE and Windows NT,
2. Aufl., O`Reilly & Associates Inc., Sebastopol, 1993.

[SHA96] Shah, R.: *Integrating Databases(JDBC)*,
erschienen in: JAVA World, Mai 1996.

[SIN97] Sinha, A. K: *Netzwerkprogrammierung unter Windows NT 4.0*
Addison-Wesley, Bonn - Reading - Melo Park u.a., 1997.

[SPE96] Spencer, K. L.; Miller, K.: *Client/Server Programming with MS Visual Basic*,
Microsoft Press, Redmond 1996.

[STE96] Stearns, D.:
Migrating Existing Information Systems to Component Architectures,
Visual Basic Group, April 1996.
http://www.microsoft.com/vbasic/techinfo/mig3tier.htm

[STR92] Stroustrup, B.: *Die C++ Programmiersprache*,
2. Aufl., Addison-Wesley, Bonn - Paris - Reading u.a., 1992.

[SUN196] Sun Microsystems Inc.: *The JAVA Door ORB*,
Sun Microsystems Inc., 1997.
http://splash.javasoft.com/JavaIDL/pages/door-orb.html

[SUN197] Sun Microsystems Inc.: *Frequently Asked Questions - Applet Security*,
Sun Microsystems Inc., 7. Juni 1997.

[SUN296] Sun Microsystems Inc.: *JAVA Development Kit 1.1 API Documentation*
Sun Microsystems Inc., 6. Dezember 1996.
http://www.javasoft.com/products/JDK/1.1/

[SUN297] Sun Microsystems Inc.: *IDL-JAVA Language Mapping*,
Sun Microsystems Inc., 1997.
http://splash.javasoft.com/products/jdk/idl/docs/idl-java.html

[SUN397] Hamilton, G.: *JavaBeans V1.01*,
Sun Microsystems Inc., 24. Juli 1997.
ftp://ftp.javasoft.com/docs/beans/beans.101.pdf

[SVE96] Back, S.; u.a.m.:
Workshop JAVA, Einführung in die Programmierung von JAVA-Applets,
International Thomson Publ. /VVA, 1996.

[TAN95] Tannenbaum, A. S.: *Verteilte Betriebssysteme*,
Prentice Hall, München - New York - London u.a., 1995.

[TAN97] Tannenbaum, A. S.: *Computernetzwerke*,
3. Aufl., Prentice Hall, München - New York - London u.a., 1997.

[VIN96] Vincent, J.: *Das Internet im Unternehmen*,
erschienen in: PC Professionell, Juli 1996.

[WIT97] Withopf, M.: *Microsoft DCOM für UNIX*,
erschienen in: iX, Jg. 8 (1997), S. 52-57.